AFIT/GCS/ENG/98M-01

Methodology for the Analysis and Design of Internet Software Components Providing Relational Database Access Through the World Wide Web

THESIS

Daniel L. DiPiro
Captain, USA

AFIT/GCS/ENG/98M-01

AFIT/GCS/98M-01

Methodology for the Analysis and Design of Internet Software Components Providing Relational Database Access Through the World Wide Web

THESIS

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Systems

Daniel L. DiPiro, B. S.

Captain, USA

March 1998

AFIT/GCS/98M-01

Methodology for the Analysis and Design of Internet Software Components Providing Relational Database Access Through the World Wide Web

THESIS

Daniel L. DiPiro
Captain, USA

Presented to the Faculty of the Graduate School of Engineering

of the Air Force Institute of Technology

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Systems

Richard A. Raines, Ph.D., Major
Committee Member

Henry B. Potoczny, Ph.D.
Committee Member

Michael L. Talbert, Ph.D., Major
Committee Chairman

Acknowledgements

I wish to thank my thesis advisor, Major Talbert, for his sound support, guidance, and much needed course corrections. I would also like to thank my committee members, Major Raines and Dr. Potoczny for additional support and guidance on both my research and my studies at AFIT.

My greatest thanks and appreciation go to my wife Gaile and my children, Danielle and Patrick. Their sacrifices have made it possible for me to pursue my studies at AFIT. They continue to be the foundation upon which I can rely in both calm and stress.

Thanks are also due to the AFIT staff and faculty, especially Angela, Carolyn and Mary Jane, that have provided more assistance, and support then this Army student was expecting or was used to. I would also like to thank my sponsor, AFIT/CI for their assistance and constructive input to my research.

Table of Contents

Table Of Figures

List of Tables

List of Trademarks

1. Apache is a registered trademark of the Apache Group

2. Common Object Request Broker Architecture (CORBA) is a registered trademark of Object Management Group

3. FormFlow is a registered trademark of Delrina Corporation.

4. Macintosh is a registered trademark of Apple Computer Inc.

5. Microsoft, ActiveX, ActiveX Server Pages, BackOffice, Distributed Component Object Model (DCOM), Internet Explorer, Internet Information Server, JScript, Microsoft Access, Microsoft SQL Server, Open Database Connectivity (ODBC), Object Linking and Embedding (OLE), Object Description Language (ODL), Personal Web Server, VBScript, Visual Basic, Visual C++, Visual J++, Win32, Windows, Windows 95, and Windows NT are registered trademarks of Microsoft Corporation.

6. Mosaic is a registered trademark of the National Center for Supercomputing Applications (NCSA)

7. Netscape, Navigator, and Communicator are registered trademarks of Netscape Communications Corporation

8. Oracle and Designer 2000 are registered trademarks of Oracle Corporation

9. Perl is a registered trademark of Larry Wall.

10. Rexx is a registered trademark of International Business Machines

11. Sun, Sun Microsystems, JDK, Java, JavaScript, JDBC, HotJava, Remote Method Invocation, and Servlet are registered trademarks of Sun Microsystems Inc.

12. Unix is a registered trademark licensed through X/Open Company Ltd.

13. VMS is a registered trademark of Digital Equipment Corporation.

Other product and company names mentioned herein may be the trademarks of their respective owners.

AFIT/GCS/ENG/98M-01

Abstract

This work examines the application of Internet software technologies to provide access to remote relational databases via the World Wide Web. The research applies these software technologies to assist the Air Force Institute of Technology Civilian Institute Program in improving operations and student-to-staff communication.

An analysis of the existing Internet software technologies revealed several competing technologies capable of performing the same database access functions. The analysis further revealed weaknesses and inconsistencies in the existing AFIT/CI database. A methodology is proposed to assist in analyzing an existing development environment and in selecting among the competing technologies to provide the web-based database access. The methodology is applied to the AFIT/CI test case to demonstrate a technique of analyzing and designing web software components that will create new and improved uses for the existing CI database. Additional recommendations are also offered to improve the existing database operations. The results of applying the methodology demonstrated that it effectively focuses the developer on the key areas of the development environment necessary to choose among competing software technologies. Additionally, the methodology was proven to be flexible in response to changes in implementation technologies.

1 Introduction.

The introduction of the Internet and World Wide Web (a.k.a. WWW or web) to the computing world are having a great impact on the traditional client-server architecture used for remote database access. The introduction of the WWW to the Internet has spawned an enormous growth in the number of personal and business computers interconnected worldwide. This has in turn provided impetus for commercial, military and educational enterprises to make their data available to a greater number of clients through this new medium. This access has been aided by the introduction of several Internet software technologies including powerful web servers and client web browser applications that have enabled the WWW to be used in a client-server role. These applications benefit the developer by reducing or eliminating the need to develop and distribute a client-side interface and have allowed clients to access multiple data sources with a single client interface (the web browser).

1.1 Problem Statement.

While the emergence of Internet software technologies has made overall improvements to the client-server architecture, the existence of many competing and incompatible standards for implementing web-based data access have made it difficult for the developer to choose a particular technology. The client is also affected by experiencing limited or no access to server and data resources that may be incompatible with their web browser software. While countless books and research are devoted to developing applications in each of these technologies, less information is available for a developer to analyze their requirements and the available Internet software technologies to choose the ones that best fit their needs.

1.2 Goals of this Research.

The goal of this research is to provide a methodology that can be used in the analysis of a

web-based data access enterprise to aid the developer in choosing the right software technologies to implement their data access services. Additionally this research will demonstrate through the use of a case study that the methodology can be used to develop Internet software components that provide a new or more efficient use of legacy relational data sources. Each major Internet software technology will be analyzed according to the functional capability it brings to the client-server architecture and its capabilities and limitations. Through the methodology, this research develops a process to analyze the existing client and server environment affecting any design decisions. It also shows a method for functionally decomposing a web software component into sub-components. Once a component is decomposed, the developer can then use the existing environmental factors and knowledge of the tradeoffs accompanying the software technologies to decide how to implement the data access provided by the component.

1.3 Case Study

To test the methodology for implementing World Wide Web (WWW) based software technology the current and desired operations of the Air Force Institute of Technology Civilian Institution program office (AFIT/CI) will be used as a test case. Specifically, this research will involve designing three Internet software components to facilitate Web-based interaction between AFIT/CI and its current and prospective students.

Annually, AFIT/CI administers the academic programs of over 2,300 students in long-term graduate programs at over 400 institutions and 3,600 professional continuing education students. Each student is assigned to a Program Managers (PMs) within AFIT/CI who oversees that students educational program. Current and prospective CI student and Program Manager interaction consists mainly of telephone conversations, paper documents sent via mail, and electronic mail. Existing data sources are currently only available in-house to the CI staff and are largely limited to a few pre-defined queries and printed reports. A few essentially isolated Web pages currently offer static information, manually updated by CI personnel as needed, to

prospective and current students.

The new Internet software components would benefit both prospective and current students by providing them with timely and accurate program-critical administrative information electronically from a central CI web site. Additionally it would improve student to PM interaction by creating several new automated functions to aid the student in not only viewing and updating their personal data, but also with the preparation of any required reports submitted to their PM. The components should also benefit the PM by minimizing the amount of time spent answering frequently asked questions and allowing him/her to receive more accurate and timely student input. The components should make maximum use of the CI database to assist the PM in managing his/her students.

1.4 Thesis Overview.

Chapter 2 provides a brief history of the client-server architecture and how the emergence of the Internet and World Wide Web has enhanced the traditional client-server model. Current Internet software technologies are discussed and compared with respect to their capabilities, limitations, and their benefits to the client-server architecture. Chapter 3 covers the methodology used to evaluate and choose the technologies necessary to implement the test case. The methodology examines the key environmental factors affecting the design of web software components and the areas in which they should influence design decisions. Additionally a method to functionally analyze components is presented to assist in determining which technologies are capable of fulfilling the requirements of a component. Chapter 4 discusses the implementation of the three Internet software components comprising the test case and finally, Chapter 5 contains conclusions and areas for further research.

2 Background

In this chapter, some background material is provided on the client-server model, the World Wide Web, and the new software technologies that the World Wide Web brings to the client-server environment. Each technology is covered with respect to its features, capabilities, pros, cons, and future impacts.

2.1 A brief history of the client-server model.

In the early days of computers, data processing was handled in a centralized manner in which all processing tasks were handled by one computer. Users communicated with the central processor through dumb terminals that allowed them to input commands to the central processor which then performed the desired database operations. These operations were typically Structured Query Language (SQL) commands or specific operations on structured data files. Results and status messages were sent back to the user on the terminal screen. Remote users would use a communications utility with terminal emulation to achieve the same connectivity as local users.

As technology improved and the number of users increased, centralized computers with many users experienced poor performance due to the processor and I/O limitations of the hardware. To fix this problem, computer architects sought ways to minimize the central processor's workload and distribute the processing tasks. The client-server model was born out of this desire for efficient distributed computing. As computers became faster, cheaper, and smaller, this model came to the forefront of applications development.

Figure 1 shows the client-server model in which the user operates a client computer, which interacts with a central server over a communication network. The server handles all access to the desired data and passes results back to the client who processes the results locally. Although the client-server model eased the workload of the server, it added new areas of concern for both

network engineers and applications programmers.

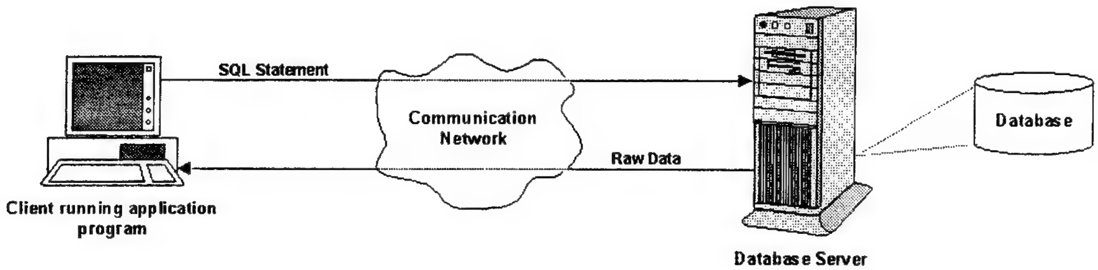

Figure 1. Client-server Architecture

In the client-server architecture, more data was now being sent over the communications network. Network designers now had to compensate for the increase in traffic over the communications links to sustain desired performance levels. Formerly, all applications resided on one machine making it easy for applications developers to manage. Now they have to deal with separate client and server applications that could reside on many different computer platforms. This new model also introduced heterogeneity in software and hardware that had previously been minimal or non-existent. The client was often using a different hardware and software platform than the server. The client possibly also resided on a different communications network architecture. This resulted in a need 'middleware' to facilitate cross-platform operations. Specific roles of middleware are the subject of subsequent sections.

2.2 Elements of the client-server model.

Although there are several variations of the client-server model, they all typically have the following in common:

a. Server(s). A central computer, which receives remote requests for information, obtains the information from a central data source, and sends results or messages back to the remote user (client). Processing that occurs on the server is often referred to as 'back-end'

processing.

b. Client. A process which interacts with one or more central servers, providing a user interface, formatting service requests, communicating requests to the back-end process, and displaying or processing end results or messages returned from the server. Processing that occurs on the client machine is often referred to as 'front-end' processing.

c. Middleware. Since even in the same enterprise there can exist hardware and software incompatibilities between the clients and server, middleware applications must be designed to bridge this gap. Middleware handles processing tasks such as user interface processing, communications with the server, and returned data processing routines. Since the hardware and software of the clients were known, the applications programmer could build the appropriate middleware to overcome any heterogeneity in hardware, software, file formats, or communications protocols.

2.3 Two vs. three-tiered database access architecture.

Client-server applications are commonly modeled using a tiered concept. The number of tiers used in client-server applications will typically be two or three. The benefits and drawbacks of a two- and three-tiered implementation for web database access are compared in the following two sections.

2.3.1 Two-tiered database access architecture

In a two-tiered architecture, the client deals directly with the application server, in this case the database server (Figure 1). All requests for data are passed to the server who sends the results back to the client for processing.

The major advantage of the two-tiered approach is that because all processing of data is done at the client, the workload of the server is kept to a minimum. This may be beneficial if the server is of limited capability, has a large workload, or if the number of clients who will be

processing large sets of retrieved data is significant.

Another benefit of this model is that it is architecturally simple. There are only two application layers, and therefore it requires minimal inter-tier communications. The simplicity reduces the potential for system failure due to communications problems between tiers.

The disadvantages of the two-tiered model include the necessity of a more complex, 'fatter', client, and increased potential for data security problems. There is no server-side processing and therefore the client must implement any functions that process the retrieved data. This 'fat' client will consume more client processor resources and occupy increased storage space on the client's machine. The potential for data security problems is increased since the client interfaces directly with the database server.

2.3.2 Three-tiered database access architecture

In a three-tiered architecture (Figure 2) the client communicates with the server only indirectly through middleware. The middleware application requests the data from the database server and may process all or part of the data returned by the server in the middle tier.

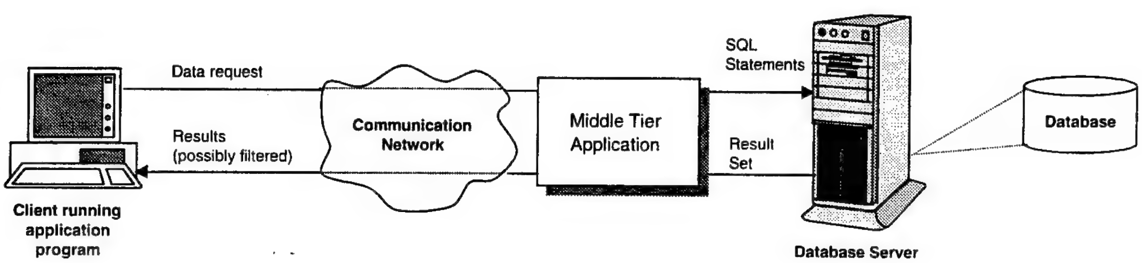

Figure 2. Three Tiered Client-Server Architecture

The advantages of the three-tiered approach are a less complex 'thin' client, more efficient utilization of communications bandwidth and increased data security. Middle-tier processing allows off loading of processing functions from the client, providing a client that is easier to design and maintain, and requires less resources from its host. This is beneficial if the client's host has limited processing or storage capability.

The middle tier will most often be placed in a location that provides a fast communication path to the data source. Middle-tier processing of the data, may reduce the results of a database operation significantly. This can result in a minimal set of results being sent back to the client. That data reduction, in conjunction with the increased communications speed from the middle tier to the data source, improves communications bandwidth utilization over the two-tier approach.

Additional data security is also obtained because all client access to data can be controlled through the middle-tier application, which can be configured to provide access through a 'firewall' if necessary. This provides increased data security to applications in which tight control over data access is required.

The disadvantages of this approach however are an increase in architectural complexity and increase in server workload. The addition of a third layer requires increased inter-tier communications to pass requests and data between the layers. This increased interaction makes a three-tiered design more difficult to create and maintain, increasing the potential for an overall system failure due to inter-tier communications problems. Additionally, performance may suffer if the number of clients accessing the services of the middle tier is large and its host computer can not adequately handle the increased client load.

2.4 Emergence of the Internet and World Wide Web.

In recent years, several forces in the computer world have led to changes in the client-server model. The process that resulted in these changes was the evolution of the Advanced Research Project Agency Network (ARPNET) into what is called the Internet today. ARPANET was the distributed communications networking backbone designed in the 1960s for use by the government in case of nuclear war. In 1990 the government separated the military portion of the network (MILNET) and decommissioned the rest, in effect giving its administration to the National Science Foundation (NSF). Until that time the network was used almost entirely for

research, education and government purposes. The NSF in 1995 announced that it would contract the administration of the network to four private companies, forming the Internet Society [Kristula97].

This process was not alone in its effect on the current state of the Internet. Three other factors have contributed to the Internet's rise to prominence in the computing world. The first is the availability of inexpensive desktop computers that are in use in both the home and workplace. In 1990 there were approximately 313,000 hosts connected to the Internet. Today there are over 15 million hosts and that number grows rapidly each year [Kristula97]. The second factor is the creation of numerous Internet service providers (ISPs) that allowed those inexpensive desktop computers to connect to the Internet at an affordable price. The third, and arguably most notable, was the development of the World Wide Web.

The World Wide Web was born out of a system created in 1990 by the European Laboratory for Particle Physics Research (CERN) as a way for physicists to share large amounts of text-based information in a distributed manner. The system made use of hypertext documents to display information. A hypertext document is a document that contains text with embedded links to other similar documents. A web user could follow links in one document to related documents or other web sites. This haphazard and deliberately unregulated cross-linking of documents led to its obvious comparison to a spider web, and its name, as it is known today.

The software that is used to view web documents, also called pages, is called a web browser. Early web browsers supported the viewing of text only documents. Led by the introduction of the first graphical web browser, Mosaic, in 1993 [Kristula97], today's browsers support a greater variety of formats. Current browsers additionally support images, audio, video, email and embedded program code. As of the time of this writing there are many versions of browsers from many vendors, notably Microsoft and Netscape. Although each version may differ slightly in what it supports, they typically all support a common baseline of functions, such as text formatting and multimedia object support.

The documents that a user views are encoded using the Hypertext Markup Language (HTML). HTML allows information to be encoded using ‘tags’, which indicate the format of the tagged item with nearly limitless variations [HTML96]. The tags allow the creator to encode text formatting, images, and links to other documents within the web page by enclosing them within a set of tags for a specific HTML option (Figure 3). The page is stored on the server and transmitted as ASCII text. The web browser of the client viewing the page, translates the HTML file and displays the information with the formatting. Figure 3 shows how a document looks when encoded in HTML. Figure 4 shows how that document looks when viewed through the users web browser.

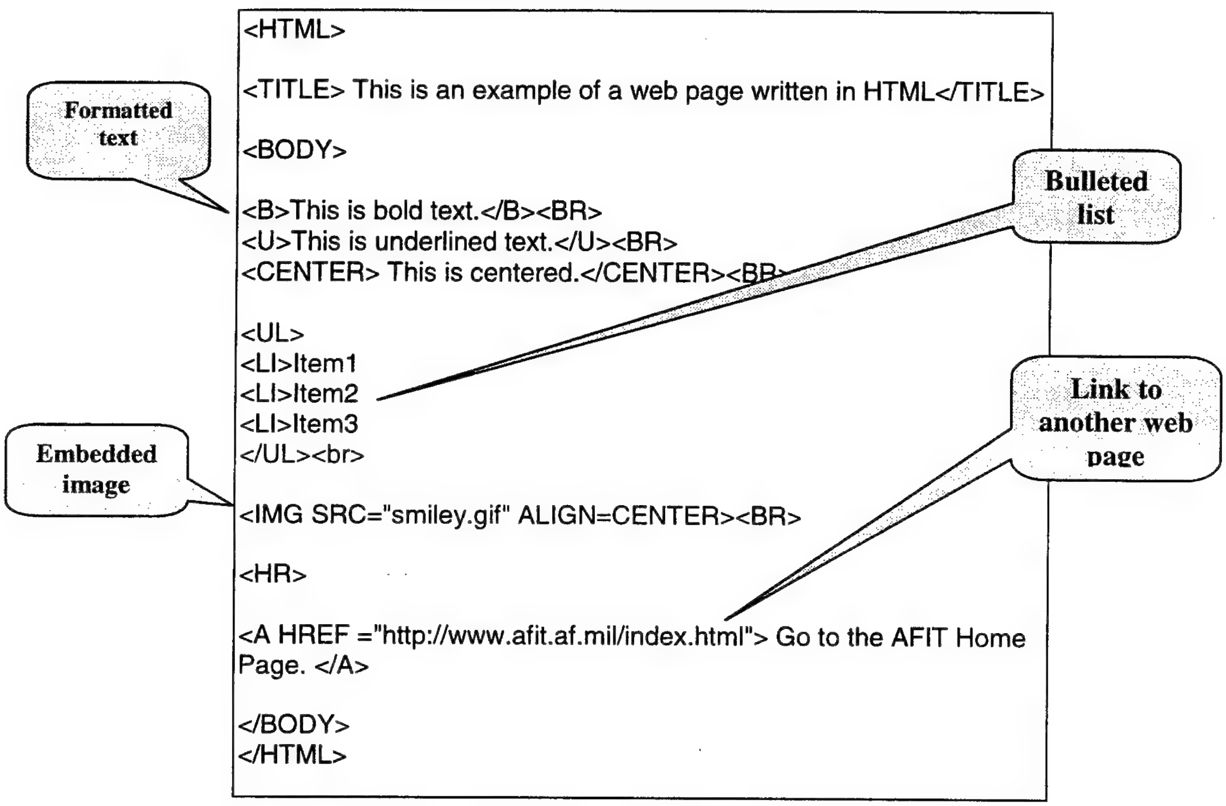

Figure 3. Example HTML Document

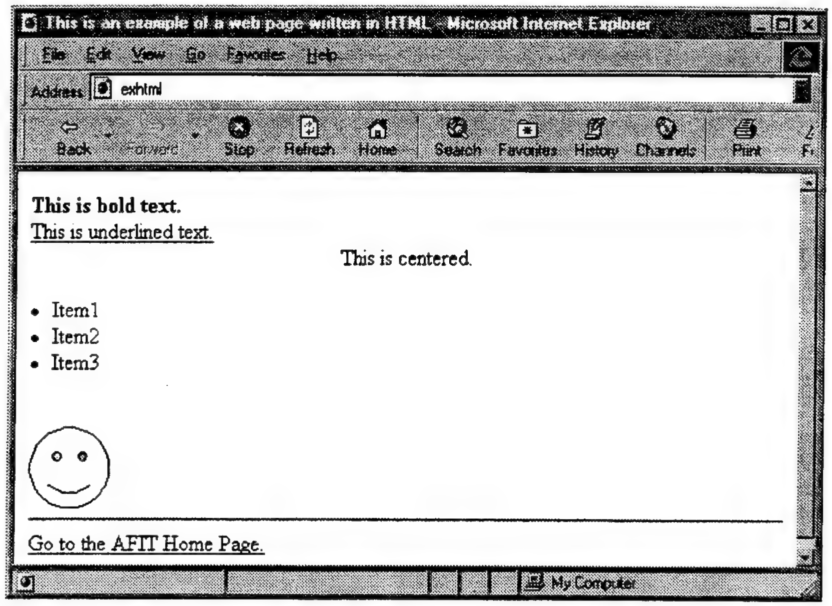

Figure 4. HTML Document in Browser Window

Not surprisingly, the protocol used to request pages from web servers is called the Hypertext Transfer Protocol, or HTTP. Requests for web pages are sent from the browser to the web server in the form of an HTTP request[1] [NWG97a]. The HTTP request contains a Uniform Resource Locator (URL) that states the protocol of the request, identifies the web server that has access to the desired HTML document, and the name of the document being requested [NWG97a]. Figure 5 shows an example URL. The URL indicates that the HTTP protocol will be used to connect to the server named "www.afit.af.mil" and request the HTML document "/mypath/index.html."

Figure 5. Example URL

---

[1] Current browsers also support other protocols such as the File Transfer Protocol (FTP)

2.5 What the World Wide Web brings to the table.

While the importance of the introduction of the Internet and World Wide Web on the computing world is well accepted, its role in the client-server architecture is still evolving. Some computer industry professionals believe for reasons such as a lack of standards for web technology, that the web is not ready to assume a full role in the client-server world [Hayes97]. Although the web's potential may still not be fully realized, it can begin to enhance the client-server environment in the following areas:

a. *Elimination of client applications development.* By using the web browser as the *universal client*, applications developers no longer have to design and deploy client side interface software. Virtually all new computers come with browser software and free or inexpensive versions for all the major hardware and software platforms are readily available. This eliminates the need to design, distribute, and maintain a proprietary user interface.

b. *Familiar client user interface.* The rise of Internet use among people today has lead to millions of people who have experience in using a web browser as a client interface. Since the client interface will be the same one they use to 'surf' the Internet at home, they will not only be familiar with how to interact with it, but will need little or no training to interface with the server.

c. *Integrated multimedia capability.* Web browsers are designed to handle multimedia data such as graphics, audio, and video. This capability makes them an ideal client to access data of many different varieties from a remote server.

d. *Downloadable compiled code components.* Web browsers have the ability to download and execute compiled code that is sent over the communication network to the user's browser. The executing code appears embedded in the requested web page. This capability allows developers to create platform-independent applications that can be sent to a client from a central server and can be run on any capable browser. This eases the burden of the developer since only one copy of the application is maintained at the source. Clients will always get the latest version

of the code when they access the Web server and request a page that contains the downloaded component. Since the code is only transmitted from the Web server and is actually executed on the client machine, it supports the client-side execution philosophy of the client-server model.

e. *Ease of scalability (client-side).* The Web client-server model is easily scaled on the client side since the intended new client has only to obtain a network connection and the readily available Web browser software. In addition, clients who only want to access the information infrequently are not forced to install any additional components, hardware or software, to connect to the server than the modem and browser that are most likely already present on their machine. The ease at which clients can connect to the Internet and use the Web allows a potentially large amount of clients to have access to a server. The number of clients that a server can handle is only limited by the server's ability to handle multiple concurrent connections, the available network bandwidth, and the speed at which the server can access the requested data. Since hardware prices have been steadily decreasing, it has become easier and cheaper to add capability to the server to handle more clients and access the data fast and efficiently. However, at this time the relative cost of network connectivity has not decreased at the rate of hardware and can be therefore affect the ability of a system to scale up as a large number of new clients are introduced.

While these factors exemplify some benefits of the web in the client-server model, several potential pitfalls associated with using the web in this fashion to access databases must also be considered.

2.6 Potential difficulties in introducing the web to the client-server environment.

The difficulties in providing web access to existing relational databases fall into three categories: deciding among competing standards, the statelessness if the web, and dealing with legacy middleware.

2.6.1 Choosing standards to use in implementing Internet access to a database

Although the HTML document format of the web is a widely accepted standard, no single standard exists for handling database access, client side processing, or networking. A number of products are put forth by competing companies such as Sun and Microsoft, who recognize that wholesale acceptance of their Internet technology could result in large profits. Consequently, it is often difficult for the Internet site developer to determine which technologies will help integrate their legacy information with the web. This is the problem for which the methodology will attempt to provide a framework, in integrating the AFIT/CI system with web access to provide more efficient use and broader availability of their data.

2.6.2 The stateless nature of the web.

A hallmark of a good database management system, to include distributed databases, is its ability to remain in a consistent predictable state. Because the web implements a request-response paradigm, it is inherently stateless. A web server maintains no persistent state information and handles each request with out any regard to previous access. This lack of persistence will concern anyone who wants to provide database access over the web but can not compromise on database consistency.

This statelessness can be overcome through several means. The first is to record state information as hidden fields in an HTML page. These hidden fields can contain information that is passed to the web server with subsequent HTTP requests [Cornell96]. The second method is by using Internet "cookies". Cookies are files that contain state information relative to a web server. The cookie is stored on the client's machine and read by the web server every time the client accesses a page. The cookie can contain user authentication information, user preferences, or other state information [NWG97b]. The third method of maintaining database consistency is by having an application embedded in a web page interface with the database directly, or through

a middle-tier application that implements some method of transaction management.

2.6.3 The web and legacy middleware.

Many current client-server applications consist of significant amounts of legacy applications code. This code may have been developed to bridge heterogeneities in database management systems, communications networks, or data representation. When introducing web access to the same sources bridged by existing middleware, the developer may need to implement bridging functions in the web application also.

This situation may be complicated by the fact that the middleware applications may be written in programming languages no longer in common use. A programmer well versed in C++ may have difficulty determining how to implement in their web applications the functions contained in legacy COBOL programs. The daunting task of addressing legacy middleware applications migration may cause some to forsake a web based access scheme altogether, or at least approach it with considerable reservation. However, the difficult task of implementing middleware functions is made easier by the inherent ability of the web to bridge many network, software and hardware differences.

2.7 Software Technologies for Web-Database Integration

Having shown how the introduction of the World Wide Web and Internet can enhance the client-server model, the focus now shifts to the software technologies that make those enhancements possible. Applications developed using these technologies fit into two categories with respect to the client-server model. The two categories contain technologies used to create code designed to run on the client (client-side) and code designed to run on the server (server-side), respectively. Within each category, sub-categories further differentiate between the technologies. This section discusses each category, the technologies that constitute them, and some key associated pros and cons of each. Additionally the most common database access and

communication techniques of Internet software technologies are also discussed.

2.7.1 Client-side technologies

Client-side technologies create code that will be executed within the client's web browser. These technologies lighten the workload of the server by performing some or all processing tasks on the client's machine. The two main types of client-side processing technologies in this category are scripting languages and downloaded compiled components.

2.7.1.1 Client-side Scripting languages

Scripting is the embedding of program code in text form, called *scripts*, within a web page. Scripts are written using one of several scripting languages that are typically a loosely typed subset of a more computationally complete programming language [JavaScript97]. A script is downloaded as text along with the HTML page that contains it. The script is then interpreted and executed in any client web browser that supports scripting.

Scripts were designed to interact with both web pages and the client's browser software. Consequently, they are commonly used both to customize web pages based on the client's browser version and to facilitate interaction with the user in an otherwise static page[2]. For example, a script may be used to validate user inputs before sending them to the server for processing.. Additionally, scripts can interact with compiled code components, such as Java applets (section 2.7.1.3), that exists on the same web page. Figure 6 shows an HTML document containing an embedded script and Figure 7 shows the results of the script's execution in the client's web browser.

[2] Often by having an image change appearance when the mouse moves over it.

```
<HTML>
<HEAD>
<TITLE>JavaScript Sample</TITLE>

<SCRIPT LANGUAGE="JavaScript">
<!--HIDE
function sayHi(field1, field2) {

   var name = field1.value
   var age  = field2.value

   if ((name == "") || (age == "")) {
               alert("You have not filled out the form correctly!")
   }

   else {
        if (age < 30) alert("Boy " + name + "! You are a youngster!" )
        else alert("Boy " + name + ", you are old!\n" + "Only " + (60-age) +
        " more years to retirement!")
   }

}
//STOP HIDING-->
</SCRIPT>

</HEAD>
<BODY>
<FORM NAME="theForm">
Enter Name Here: <INPUT TYPE="text" NAME = "field1" VALUE = ""><BR>
Enter Age Here : <INPUT TYPE="text" NAME = "field2" VALUE = ""><BR>
<Input TYPE="button" VALUE="Process"
 onClick="sayHi(theForm.field1, theForm.field2)">
 <INPUT TYPE="reset" VALUE="Reset Form">
</FORM>
</BODY>
</HTML>
```

JavaScript code

HTML input form

Pushing button in the form calls the script

Figure 6. JavaScript Code Example

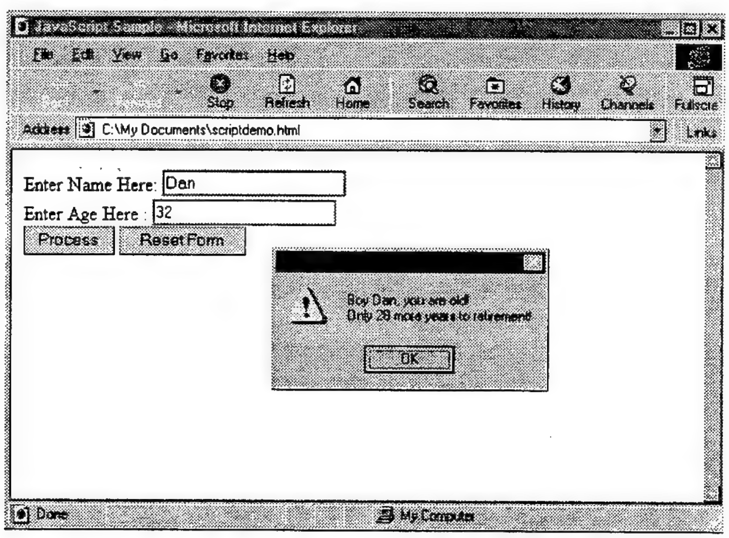

Figure 7. JavaScript Execution in Client Browser

Some key benefits and limitations of using client-side scripting and a brief description of the three most widely used scripting languages are examined below.

Advantages of scripting.

Scripts achieve platform independence in a heterogeneous environment by being interpreted and executed locally in each client's browser. By interacting with elements of the HTML page and the web browser software, a script can add dynamic behavior to an otherwise static web page. More importantly, however, because scripts execute in the client's browser, script-based processing decreases not only the burden on the server, but also network communications between the client and server.

The reduction of server workload is achieved in part by allowing the scripts on the client to do relatively simple computations and processing on user entered data before it is sent to the server. Consider, for example, a script being used to validate user input to an HTML form. When an HTML form is used to obtain user input, the input is typically transmitted to the server for validation. Any data entry errors would then have to be transmitted back to the client with an appropriate error message so the client could make corrections for resubmission. By using a script to check the input form, network usage can be reduced by checking the data for correctness before it is sent to the server for processing. The client will get rapid feedback and the server can be assured that any processing that it does will be on valid data [Mueller97].

Limitations of client-side scripting

Scripting languages are not computationally complete. Further, although they implement some variation of an object model, they are not object-oriented and so lack support for complex

data types, inheritance and polymorphism [JavaScript97] [3]. Consequently, they are limited by their minimal expressive ability to model real-world entities, and by their relative computational weakness as described above.

Additionally, applications written in a scripting language can not be used to establish network connections or database access and therefore must work in conjunction with other technologies to accomplish these tasks. Another possible disadvantage of scripts is that by being embedded in the HTML page as ASCII text, they are completely visible to the user on demand[4]. As a consequence scripting may not be a desirable choice if the task that the script is to perform contains any proprietary or formulas or sensitive information.

In light of these limitations, any sophisticated client-side processing will require the use of more computationally complete languages such as Java. It follows that in a business environment scripting will for the present play a supporting role, targeting a specific subset of client-side processing tasks.

Scripting languages

The three most widely used scripting languages are JavaScript, JScript, and VBScript. JavaScript and JScript are almost identical and are both based on the very powerful Java programming language [Gesing97]. Microsoft's JScript however does not support some elements of JavaScript and introduces others only supported in Microsoft's own web browser [Gesing97]. Because of this, an application written using JScript or JavaScript may not be completely platform independent.

VBScript is based on the widely used Microsoft Visual Basic (VB). It is the built-in scripting language for many Microsoft products, and thus serves to lower the learning curve for developers who are familiar with it. Although it is an interpreted language like the other client-

[3] Object based vs. Object Oriented
[4] Web browsers allow the user to view the source HTML if desired

side scripting languages, VBScript is an expressively ricer and more computationally complete language. However, the main drawback of VBScript is that when used on the client-side, it is only supported in Microsoft's web browser and therefore not very portable to non-Microsoft web browsers [Mueller97].

2.7.1.2 Downloaded compiled components.

This category of client-side software technologies consists of code that is compiled and stored on the server. When referenced in a web page, the compiled code is downloaded to the client's computer where it is then executed in their web browser. The code runs in the web browser's run-time environment (virtual machine) which has the capability to execute the application. The run-time environment can be built in to the browser or implemented via a browser software plug-in. For example, Microsoft's Internet Explorer browser includes an embedded run-time environment for two basic types of components, applets created using Sun Microsystems' Java programming language, and Microsoft ActiveX controls, which can be written in a variety of programming languages. Netscape's Communicator, however, needs a browser plug-in to handle ActiveX components.

These compiled component applications are written in computationally complete programming languages such as Java, or C++. They achieve platform independence by being compiled at the server into an intermediate executable form that lacks any platform specific bindings. The web browser of the client, through the run-time environment, provides the binding to the client's platform at execution time. The power and portability of these languages make it possible for them to be used in a limitless variety of applications. Currently, most applications of this type are written to perform complex processing tasks such as animation, database access, network communications, graphics processing, etc. Another capability of these types of components is that when executing within the browser, the public methods of these components can be called from a client-side script executing in the same browser window. The capabilities of

downloaded platform independent code are continuing to evolve in both industry and academia through the Network Computer (NC) concept.

A NC has a fast processor and a large amount of RAM, but no mass storage. When a NC user needs an application, it is downloaded from an application server and run in the client's computer. When the application is no longer in use on the client machine, it is discarded. While this concept will likely become part of mainstream computing in the future, the current cost of network connectivity is preventing it from reaching its full potential for large applications. For example a fifteen-megabyte application that would download quickly over a 100 MBPS fiber optic link, would not achieve satisfactory download times on a typical 10Mbps LAN or 28.8 KBPS dial-up connection. However, if network technology continues to advance, and connectivity costs decrease, the NC will become a viable and cost effective technology for client-side use. In the mean time developers should remain concerned with the size and download speed of any compiled components [Hamilton96].

The remainder of this section discusses the current benefits and limitations of downloaded compiled components and the two most widely used types, which are Java applets and ActiveX controls.

Advantages of downloaded complied code

Unlike the client-side scripts discussed earlier, the languages used to create compiled components are much more computationally complete. Their ability to easily implement network and database access and process multimedia data make them a robust client-side development tool. Additionally, because the code is executed on the client's computer, the reduced processing load on the server allows the server to provide the same application to a greater number of clients without a significant increase in server-side processing.

Platform independence, central storage of the compiled code, and automatic downloading of the components gives the developer the capability to more rapidly create, maintain and distribute client-side applications. The client benefits by always receiving a current version of the application and the elimination of having to store any applications code locally. These attributes of downloaded compiled code provide a previously nonexistent capability to the client-server model and makes applications created in this manner well suited for use on the World Wide Web and the Internet [Morrison97].

Limitations of downloaded compiled code.

Applications developed as downloaded compiled code will typically be more complex and potentially of significantly greater size than static HTML pages or client-side scripts. Also, there is no limit on the size of a downloaded code component. However, an applet that takes an excessive amount of time to download to the client's machine will not provide adequate response time for the user. Additionally because each browser version implements the run-time environment for components differently, components may function correctly in some browsers and not in others [Mueller97]. The addition of these run-time environments in the browser also adds to the complexity and size of the browser software as well. While not typically an issue for the developer or most clients, if there is limited disk space on the client's machine, the increased browser size may present a problem.

Because a downloaded component is executed on the client's computer the potential exists for the application to inadvertently, or deliberately, harm their system. In light of this, the run-time environments implement a varying degree of security constraints on any compiled components executing in the browser. For example, the Java run-time environment will initiate a security exception if an applet attempts to access the client's hard disk or printer, or tries to

communicate with any machine other than the server [Morrison97][5]. Conversely, ActiveX controls function as a native application and can therefore completely interact with the client machine to include printing and file I/O [Mueller97][6].

Contemporary compiled component types

Compiled code designed to download and run in the client's browser comes in two main forms. The first is a called an applet. Applets are written using one of several APIs in Sun Microsystems' Java programming language, a powerful object-oriented language modeled after C++. Some elements of C++, such as pointers and multiple inheritance were omitted from Java for simplicity and security. However, through a comprehensive but easy to use set of API's and built-in complex data structures, Java simplifies many complex tasks such as database access and network communications. Java applets are compiled into machine-independent byte-code that can be downloaded automatically and run in any browser that supports a Java run-time environment (JRE) [Mueller97]. Although some browsers do not support all elements of the Java language, most implement a JRE that supports the majority of the languages functions. This wide support for Java in many browsers enhances the portability of any applications written with the language.

The second type of compiled component is an ActiveX control. ActiveX is Microsoft's standard application development component technology[7]. Like Java components, ActiveX controls can establish network connections and access databases within native code. ActiveX is based on Microsoft's Object Linking and Embedding architecture (OLE) which was designed to let windows applications share data. ActiveX controls are compiled at the server into a proprietary intermediate form called an OLE Control Extension (OCX) which is executed in the

[5] Some browsers offer the option of turning the security constraints off. Others allow code to be authenticated via a certificate before performing potentially unsafe tasks.
[6] This can be seen as a benefit or a risk, depending on the application and the clients.

browser's run-time environment, much like Java byte-code. An OCX file is an extension of a Windows Dynamic Link Library (DLL) that has a data transfer capability. This capability allows ActiveX controls to communicate with other ActiveX controls and other OLE objects [Mueller97]. ActiveX controls can be created using any tool or language compiler that can produce an OCX file. Because ActiveX is based on OLE, it benefits from a tight coupling with the Microsoft Windows 32-bit operating system. This tight coupling allows the ActiveX control to look like a native Windows application and run at native speed [Chappel97]. However, ActiveX controls can only run in a browser that has an ActiveX virtual machine (run-time environment). Currently only browsers that run on the Microsoft Windows platform have the required virtual machine and therefore ActiveX controls are not portable to browsers that run in other operating systems [Harmon97]. Another limitation of ActiveX controls is that they must be invoked using a scripting language such as JavaScript or VBScript. Java applets on the other hand require no use of scripts.

2.7.2 Server-side technologies

Server side technologies create code that will be executed on a server. The server can be a web server, a database server, or an application server. Applications created in this fashion assume tasks that can be done more efficiently or securely on the server than on the client. Additionally server-side technologies can create applications for the middle tiers of a multi-tiered architecture. The two main types of server-side technologies are stand-alone applications and web-server processing.

2.7.2.1 Stand-alone applications

Stand-alone applications are independent programs that are run on the server-side of the client-server architecture. They are independent of the web server software and can be written in

[7] ActiveX components can be used to build both Internet and non-Internet applications

any object-oriented or non-object oriented language supported by the server's hardware platform and operating system platform. In a web-based client-server environment, stand-alone applications will typically perform middle-tier or back-end functions, such as database access.

These stand-alone applications will communicate to any client-side applications, other stand-alone programs and data sources through one of the communications methods discussed in Section 2.7.3.

Advantages of stand-alone applications

Stand-alone applications run independently on a server at native operating system speeds, and are tightly bound to the platform on which they run. Because they are a running process on the server, they do not cause any performance loss due to downloading or start-up as in the case of downloaded compiled code that is run on the client's machine. They can also perform an unlimited number of middle-tier or back-end tasks because they can be written in full computationally complete programming languages.

Their computational power allows them to completely interact with the computer on which they are executing. They have the maximum allowable access to any system resources including file and printer I/O and network communications. This overall tight coupling to the system they run on allows them to be optimally tuned for both execution speed and database or network access.

Limitations of stand-alone applications

With the exception of Java programs, the tight binding of stand-alone applications to their system platform negates any application portability. They must be rewritten and recompiled for each different platform. An exception to this includes Java applications which, like applets, are compiled into platform independent byte-code that is executed in a run-time environment on the server [Morrison97]. Sun Microsystems has developed a Java run-time environment for most

hardware and software platforms. Although the run-time environment adds some overhead in binding the byte-code to the run-time platform, the portability of the applications makes it an attractive option for developers. Additionally, stand-alone applications do not communicate using the HTTP protocol and therefore will have to communicate with the user through another component that has that capability[8].

2.7.2.2 Web-server processing

Web-server processing consists of applications that are either spawned from the web-server software as an independent process or executed within the web-server itself. Applications that are spawned from the web server can be written in any programming language that can read and write to the standard I/O. This interaction through the standard I/O is referred to as the Common Gateway Interface, or CGI, which will be discussed later in this section. Applications that run in the web-server process can also be written in many programming languages, but unlike CGI applications, are treated by the web server software as native code within its main process. This eliminates the need to pass inter-process data through the standard I/O, which is much slower then passing data within a single process [JavaSoft97c].

Web server applications are invoked by the web server in response to a client's HTTP request to run that application. Results are returned to the client as an HTTP response containing an HTML document (web page).

HTTP requests sent to a web server contain information about the identity and platform of the client. A web server process often uses this information to dynamically provide a web page to the client that is customized for them based on the client's request. They also are used to receive and process client queries entered through an HTML form. The web server process executes the query on the data source and returns the results through the web server to the client.

[8] The exception to this is an application that implements a web-server.

The remainder of this section describes the advantages and limitations of using web server processing and three prevalent forms of this technique.

Advantages of using web server processing.

Web server processes not only interface with the client through the web server, but also exist as application code running on the server. They can take advantage of both to implement several unique functions. They can be used to return HTML documents to the client that are generated dynamically based either on the client's HTTP request or the results of some processing done on the clients behalf. Because HTML is browser independent, a web server process can service virtually any client without concern for compatibility. This is very useful if the developer is unsure of what browser version clients use or if cross-platform compatibility is essential. Additionally, because the client receives any results as HTML they can be easily saved to disk or printed by using the inherent capabilities of the browser software.

Web server processes also have the ability to parse HTTP requests for information regarding the client making the request. The information contained in the HTTP request can inform the web server process of who the client is, the vendor and version of the client's web browser, and many other characteristics of the client making the request. A developer can use this information to guarantee service to a wider variety of clients by delivering customized components.

The ability to dynamically generate HTML code also allows the developer to service the client with current, possibly near real-time, data. This reduces the amount of static web pages that have to be written and stored on the web server. Network communications between the client and server are also reduced because the client receives only the information needed instead of a static document that might contain large amounts extraneous data.

By running as an application on the server, like stand-alone applications, web server processes can access all file, printing, and network resources of the server. They can use this

capability to perform functions such as establishing network communications to stand alone applications, providing database access or logging client transactions to persistent storage for security or recovery purposes. Additionally, because all access to the application is through the web server, the developer will be assured more data security than with the method of using downloaded components. This is because the web server hides the details of any database access or processing, and the client is left with the abstraction of merely requesting and receiving HTML documents. Additionally, by providing the only access to the applications, the security access controls integral to most web servers can now be used to control access to the web server processes also.

Limitations of web server processing

Web server processes are executed from the web server and therefore every access to the application will use one logical connection on the server. If the number of clients concurrently using a web server process is large, the web server may become over burdened. These connections stay open until any processing is complete and the user is sent a response. If the web server application is querying a large database, a connection may be open for quite some time. As a result, the client's response time will rise in direct proportion to the number of current connections being handled by the web server.

Independently spawned web server processes also suffer similar performance loss. In this case it is due to both the significant context switching needed to start the new process and the latency inherent in communication through the standard I/O. Figure 8 shows the cost of spawning an independent process compared to one run as part of the web server.

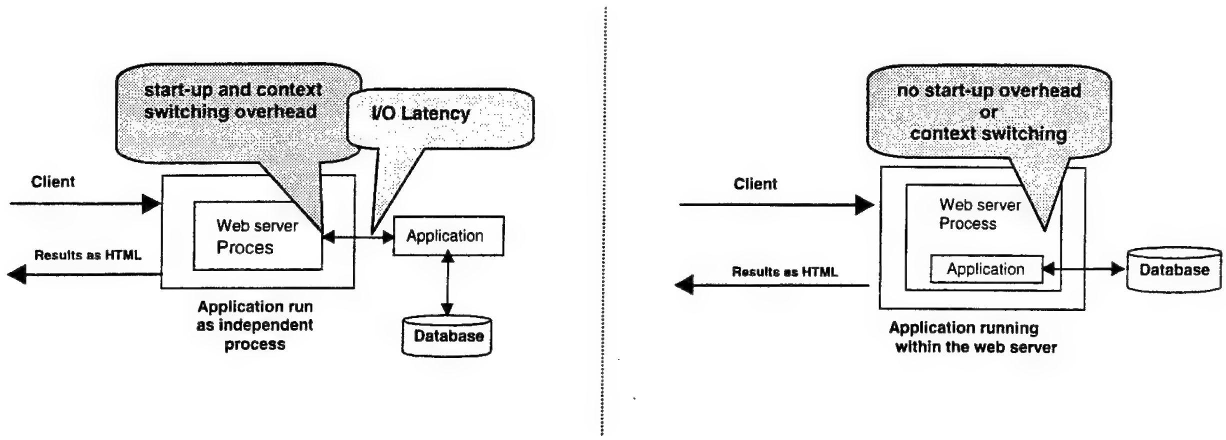

Figure 8. Comparison of web server processing techniques

Web server processes are also limited in their client interface capabilities. A web server process is limited to output formats supported by the web server and thus can only return an HTTP response containing a web page. Consequently, they can not directly interface with any client-side compiled code or script.

Current implementations of web server processing

This section discusses three of the most widely used implementations of web server processing, CGI applications, Java servlets, and Microsoft ActiveX Server Pages (ASP).

CGI does not refer to a particular language. Instead it is a class of program or script that can be invoked by and provide results to a web server. Currently, the most common language to implement a CGI application is Perl, but as stated previously, a CGI application can be written in virtually any programming language [NCSA97]. CGI applications are widely used to produce dynamic HTML pages based on the client's input. This input is usually entered into an HTML form. CGI has been widely used because most web servers include support for it; until 1997, it

was the only available technique for web server processing[9]. Because CGI applications are spawned as a separate process, they suffer from the processing overhead costs discussed above.

There have been two advances in the area of CGI which have eliminated some of its associated overhead. The first is the use of Microsoft Internet Server API (ISAPI). ISAPI is a Windows API for writing server-side extensions to a Windows-based web server. It allows CGI applications to be compiled into a dynamic-link library (DLL) which is allowed to run within the web server process. The drawback to ISAPI, however, is that the code contained in the DLL file must be written in C++, and ISAPI only works on the Windows 32-bit architecture [Perl97]. This makes ISAPI applications not portable to non-Windows servers. The second advance is called 'Fast CGI'. Fast CGI improves CGI performance by spawning a single process for all requests instead of one per request. This reduces the start-up and context switching overhead normally associated with CGI.

The second major implementation of web server processing is the use of Java servlets. Servlets are compiled Java programs that use the Java Servlet API to interface directly with the web server to provide the same functionality as a CGI application [JavaSoft97c]. Because a servlet is run within the web server process, it does not suffer the context switching and I/O costs of standard CGI. Servlets are designed primarily to run in a Java-enabled web server, but Sun also provides software patches that allow servlets to run, similar to ISAPI, in non-Java web servers. The inherent computational power of Java, its useful APIs for networking and database access, and its inherent portability make servlets an enticing option for developing applications involving web server processing [JavaSoft97c].

The last implementation of web server processing to be discussed is ActiveX Server Pages (ASP). Microsoft created ActiveX Server Pages as the integral server processing component of their web server called Internet Information Server (IIS)[Microsoft97a]. Although

[9] In 1997 Sun introduced Java servlets [JavaSoft97c] and Microsoft introduced ActiveX Server Pages [Microsoft97a]

IIS also supports traditional CGI through the standard I/O, ASP applications, like servlets, run within the web server process to maximize performance and minimize client response time. ActiveX Server Pages can be written in VBScript and JScript and plug-in extensions provide support for two other popular scripting languages, REXX and Perl [Microsoft97a]. ActiveX Server Pages can also invoke compiled server objects to perform complex tasks such as database access and email transfers. If the enterprise providing web access to databases is committed to using Microsoft products for back-end services, ASP will be a strong option to provide services to the front-end clients.

2.7.3 Web component communication techniques

Having now covered the main technologies that can be used to create web applications, the techniques available to allow these distributed applications to communicate will now be briefly discussed. The distributed nature of web applications requires them to have the capability to interact with other applications in order to accomplish some overall task. These mechanisms or techniques use the Transmission Control Protocol / Internet Protocol (TCP/IP) as their basis for inter application communications. This is primarily due to the use of TCP/IP as the internetworking communications protocol of the Internet and World Wide Web. The techniques to be discussed include the use of TCP/IP sockets, Sun Microsystems' Remote Method Invocation (RMI), Microsoft's Distributed Component Object Model (DCOM), and the Common Object Request Broker Architecture (CORBA).

2.7.3.1 TCP/IP Sockets

The use of TCP/IP sockets is the lowest level of communications available to the web programmer. TCP/IP connections make use of an Internet Protocol address (IP address) and a logical port number. The IP address is a unique address assigned to an individual computer. The

port number is logically assigned by the network operating system of that computer to identify individual applications or processes that require the use of the network connection. The use of logical ports allows more than one application to share a single network connection. This addressing scheme can be thought of in terms of a large office telephone system. The IP address represents the main telephone number of a company and the port an extension for a specific employee.

Within a computer, a TCP/IP socket is established on a unique IP address - port combination. An application can create two types of sockets, inbound sockets and outbound sockets. Applications create inbound sockets to listen on a particular port for requests to communicate from other applications. Outbound sockets are used by an application to call other applications that are listening for connection requests. Therefore, if an application wants to both originate and receive connections, it must implement both types of sockets. In order to create a connection to an application's inbound socket, the calling application must know the IP address (or host name[10]) of the destination and the port on which it is listening.

A connection request contains the IP address and port number of the destination and the IP address and outbound port number of the sender. This gives the destination the necessary information to complete the full-duplex connection between applications that allows it to transmit data back to the sender. Once the connection is established, the two applications can exchange data over a mutually agreed upon protocol implemented in both applications [Saadawi94]. While the other technologies discussed in this section use sockets as their underlying communications technique, unlike sockets, they all have a well-defined protocol to exchange data. This provides the programmer a level of abstraction above specific details of establishing the connection.

---

[10] Name server processes can map the host name of a computer, i.e. "www.nobody.com", to an IP address.

Advantages of using sockets

While sockets provide the lowest level of abstraction from communications details, they can be implemented in most Internet software technologies. Because most high level programming languages support the use of sockets, they can be implemented in downloaded compiled code components, stand-alone applications, and web server applications. This cross platform compatibility creates a common method in which to interconnect applications created in different technologies that may not support any higher-level means of communication. For example there is no high level protocol allowing a stand-alone application written in Fortran, running on a Unix computer, to communicate with a Java applet running on an MS Windows PC. Any communications between the two can however be achieved via a socket connection.

Limitations of using sockets

The low level of abstraction provided by using sockets means that the programmer must not only be aware of specific details regarding a network connection, but also must implement all of the aspects of the connection in the application being written. For example, when using sockets to communicate between applications, the programmer must manage the creation, destruction and monitoring of all sockets required by their application. Also, because a computer may have many processes that require use of the network ports, the programmer must additionally ensure that no conflict over port usage arises within competing applications.

Additionally, the programmer must also develop and implement the protocol to marshal data between the source and destination computers. Because different programmers may write the source and destination applications, an ill-defined or improperly implemented protocol could lead to communications errors from protocol mismatch.

Although using sockets may be the only way to connect applications of different types, the requirement of managing all aspects of a connection will increase to the complexity of any design using it to communicate.

2.7.3.2 Remote Method Invocation.

Remote Method Invocation (RMI) is a Java programming language API designed to allow Java applications to invoke operations on remote (distributed) objects also written in Java. It is the native remote-object management tool of Java that allows programmers to create distributed Java-to-Java applications [JavaSoft97b]. Using RMI the methods of remote objects can be invoked from another Java application on the same computer or over a network. In effect, it is similar to a C++ remote procedure call. RMI uses a technique called object serialization to marshal and un-marshal method parameters to and from the remote object. Serialization allows objects to be passed in their original form across a data stream (socket connection).

While RMI uses sockets to create connections, the level of abstraction it provides, it removes the programmer from having to know the specific details of any connections. The programmer creates a remote interface to any remote objects. He then registers those objects with a registry that is run on the computer owning the object. This process makes the method available to remote clients allowing them to reference and invoke it the same as a native method. The need for complicated data exchange protocols is eliminated because the exchange is now done through parameter passing between the application and the remote method [JavaSoft97b].

Advantages of RMI

RMI's strength lies in the fact that it is an extension of the core Java language and as such will be easier and more intuitive to implement, between communicating Java applications [Curtis97]. Also RMI is a less complex because it uses one language (Java) for both the object and interface definition.

The chief advantages of using RMI are that method parameters are serialized to the remote object and the details of the connection are taken care of for the programmer. Therefore no network sockets or protocols need to be set up by the client to invoke the remote method on a

server. Additionally because RMI uses the Java language, its use within a Java application is simpler that that of DCOM or CORBA which may have one language for the interface and another for the application.

Limitations of RMI

There are several drawbacks to RMI. The first is that RMI is a Java-only API and therefore can only be used to interconnect two applications written in Java as stated earlier. Also, by not providing directory or other services, RMI provides less power and functionality than CORBA, which provides these services and more. Because Java applications must also run in a virtual machine, applications that use RMI may have worse performance than similar CORBA or DCOM applications [Morgan97].

Additionally, portability of RMI applications to Microsoft Internet Explorer clients is limited, if possible, because the only web browsers currently supporting RMI are Sun's HotJava and Netscape's Communicator.[11]

2.7.3.3 Distributed Component Object Model

Microsoft's Distributed Component Object Model, or DCOM, provides a standard binary interface in which distributed objects, written in any language, can make remote procedure calls to other similar objects. DCOM allows distributed objects including ActiveX controls to communicate, and to be logically combined to create larger applications. DCOM objects communicate through remote procedure calls over TCP/IP sockets. DCOM is language independent, and although an object can be written in any language, Microsoft's Object Description Language (ODL) is used to define the interface to an object [Morgan97].

Unlike RMI, which is a top-down design, DCOM was built from the bottom up, not as a new technology, but rather an extension of Microsoft's existing Component Object Model

[11] Microsoft has stated that they will not support RMI because they believe DCOM is better.

(COM). COM, in turn, is based on Microsoft's Object Linking and Embedding (OLE) technology. Although DCOM implementations are being ported to other platforms for future release, DCOM is currently an option for the Windows 32-bit environment only [Morgan97].

Advantages of DCOM

The most noticeable advantage of the DCOM model is its tight integration with a WIN 32 operating system, such as Windows NT or Windows 95. This gives an application written using the DCOM model a more native look and functionality when used in a Windows environment. As discussed previously, this allows an application using DCOM (ActiveX control) to function as native code. DCOM objects can have access to all native functions such as printing, memory, and file IO. DCOM is and will continue to be an integral part of Microsoft's Win32 architecture and therefore important to anyone creating ActiveX controls or server-side applications for a Windows platform.

Limitations of DCOM

DCOM's greatest drawback is its limited platform support. By extension, since object interfaces are defined in a proprietary language, ODL, applications using DCOM and DCOM objects not very portable, either.

Another criticisms and possible disadvantage of applications using DCOM is that DCOM applications (ActiveX controls) have access to native operating system functions, such as file IO and memory. Their added power comes with the greater potential security threat of allowing remote code to interact with the system at that level. Hostile DCOM objects could easily snoop around in a client's PC, insert or delete files, corrupt system memory, or otherwise behave maliciously [Mueller97].

2.7.3.4 The Common Object Request Broker Architecture

The Common Object Request Broker Architecture, or CORBA, is neither a software package, nor a programming language, but an inter-object communications standard developed by the Object Management Group (OMG)[12] [OMG97a]. CORBA is designed to allow applications to communicate with one another regardless of the language in which they are written, or the software and hardware platform on which they exist. This communication is facilitated through an Object Request Broker (ORB).

CORBA is defined by the OMG as:

> ..the middleware that establishes the client/server relationships between objects. Using an ORB, a client can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call and is responsible for finding an object that can implement the request, pass it the parameters, invoke its method, and return the results. The client does not have to be aware of where the object is located, its programming language, its operating system, or any other system aspects that are not part of an object's interface. In so doing, the ORB provides interoperability between applications on different machines in heterogeneous distributed environments and seamlessly interconnects multiple object systems [OMG97a].

Figure 9 shows how a client object can use the ORB to invoke the methods of a remote object registered with an ORB [Acker97]. In addition to an ORB, directory and other services, CORBA also provides an interface definition language (IDL) to define an object's interface.

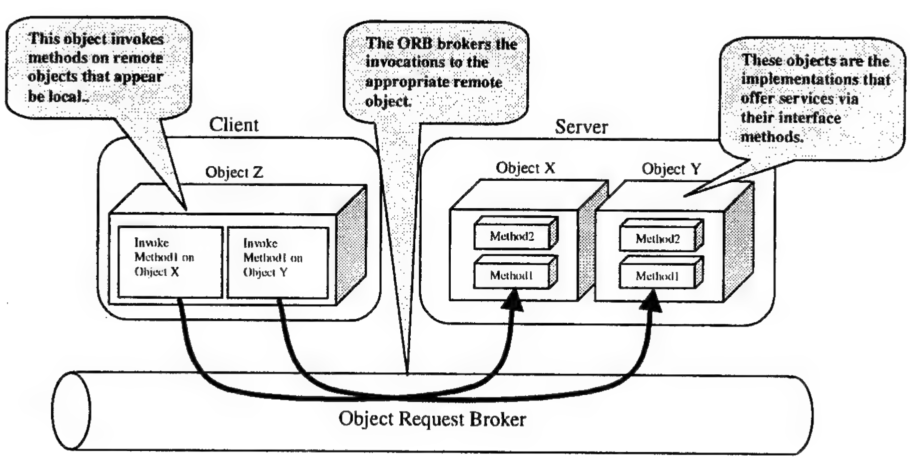

Figure 9. Invoking a method on a remote object via an ORB.

---

[12] The OMG home page is www.omg.org

Advantages of CORBA

The greatest advantage of CORBA is that it promotes a standardized object interface language, and a platform-independent way to access and use distributed objects (methods). These features make CORBA a portable, flexible, and scaleable architecture to use in implementing a distributed system.

Disadvantages

While the CORBA standard is not new, it has not been wholly embraced even by some OMG corporate members such as Sun Microsystems or Microsoft [OMG97b]. The appearance of Sun's RMI and Microsoft's DCOM model show that major software developers are not quite ready to use CORBA for their native remote object communications. Further, even existing CORBA-based products differ in their level of CORBA compliance [Orfali97].

CORBA applications are also quite complex and often require additional code libraries to operate. While this might not be a problem for computers with adequate processing and network connection resources, downloading a large CORBA application as an applet or ActiveX control might be unacceptable to a web client. The results in [Acker97] provide a clear example, where download times increased significantly for CORBA implementation over non-CORBA implementations.

The complexity, lack of wholesale acceptance, and dubious standardization enforcement of CORBA is causing some enterprises carefully consider any use of CORBA, if at all. Pountan and Montgomery, in their article titled *Web Components* cite the time it took CORBA to become a workable standard, and it's complexity as the main reason most developers are settling for a less complex solution based on ActiveX or Java [Pountan97].

The current complexity of CORBA will most likely ensure that it remains in the domain of large corporate servers for the near term. However, the fact that Netscape has included ORB features in its latest browser is very promising for even small-scale developers. If more browsers

include the capability to use ORB services, the need to download large libraries of code to handle those features will be eliminated in applets and ActiveX controls. This will be a major step in helping CORBA reach its full potential as a true cross-platform solution [Baker97].

2.7.4 Database access techniques.

What makes the Internet software technologies discussed in this section more than just a means to liven up a static web page is their ability to interact with relational database management systems. Web-based data access can be achieved through several methods. While some methods may be supported in several technologies, most are only supported in one technology. No method is necessarily better or worst than any other, however, the developer's choice to use a particular technology to implement components may require the use of a particular method of database access also. This section will discuss the use of embedded SQL, Sun Microsystems' JDBC API, and Microsoft's ActiveX Data Objects (ADO).

Embedded SQL

For years developers have had the ability to embed database access functions in software applications. Many software compilers can be purchased with pre-compilers that allow a software developer to include database access code (SQL), which is declarative, within a procedural language such as Pascal, C, or Ada. The developer could then invoke the compiled code from a web page as a CGI application. While this technique is more complicated than the others to be discussed, if the back-end platform (server) does no support other methods of database access, this may be the only option available to the developer. Figure 10 shows a small example of SQL commands embedded in Ada source code.

```
.
.
.
Procedure ada_db_access is

Item : string(1..10); Cost : float;

-- Declare the cursor to read the table
EXEC SQL DECLARE test_Cursor CURSOR FOR

     SELECT ITEM_NAME, COST FROM TEST_DB WHERE COST =
29.00;

begin

   -- Logon
   EXEC SQL CONNECT : "user/password";

  -- Execute the SELECT
  EXEC SQL OPEN test_Cursor;

  -- This is the FETCH loop

  PUT_LINE("ITEM      COST");
  PUT_LINE("--------  ------"); New_Line;

  FETCH_LOOP: loop

    BEGIN
      EXEC SQL FETCH  test_Cursor INTO :Item, :Cost;
      PUT(Item); Put("     "); Put(Cost); New_Line();
    END;
  END LOOP FETCH_LOOP;

  -- Disable cursor
  EXEC SQL CLOSE test_Cursor

  -- commit changes
  EXEC SQL COMMIT RELEASE;
  .
  .
  .
```

Figure 10. Embedded SQL example

Java Database Connectivity (JDBC)

JDBC is a database access API, developed by Sun Microsystems Inc, that allows Java programs, both applets and applications, to access relational databases [JavaSoft97a]. JDBC allows an application to interface with a relational database through a separate software driver that provides the link between JDBC and that particular database management system. Sun provides a driver to link JDBC with databases that support Microsoft's Open Database Connectivity protocol (ODBC) with the standard Java package [Microsoft97b]. This driver gives

a developer the ability to access the many database products that have an ODBC interface. Other database management vendors are responsible for the development of any drivers to access their product directly or on non-Microsoft platforms [JavaSoft97a].

Unlike embedded SQL programs, applications that use JDBC are completely procedural and contain no declarative SQL commands. SQL queries are sent as a parameter to a method call. The method then executes the query and returns a result set or any applicable error messages. Because it is implemented in Java, JDBC can be used on the server-side within stand-alone applications or web server processes, and can also be used in downloaded client-side applets. The only limitation of JDBC is that it is a Java-only API.

Active Data Objects (ADO)

Active Data Objects are compiled database interface applications residing on a Microsoft Windows platform that can be invoked by stand-alone applications and ActiveX Server Pages (web server process) [Microsoft97a]. ADO provides the same database access capabilities as JDBC. However, ADO will interface only with a database that can be accessed via Microsoft's ODBC protocol. While this includes a number of popular and powerful databases, the requirement to interface through ODBC limits ADO's use to Windows platforms. Applications written to use ADO are written in programming languages that support the Microsoft platform such as Visual C++, or Visual Basic. Because ActiveX Server Pages are also tied to the Microsoft platform, the use of ADO in web server processes is similarly limited to Windows platforms.

The main benefit of ADO is for developers who will use Microsoft products to perform back-end processing (database and web-server). The tight integration of ADO to the Microsoft architecture makes it very easy and reliable to access relational data using this method.

2.7.5 Summary

This chapter presents an overview of the client-server model with a focus on how the introduction of the Internet and Web has changed the client-server model's fundamental character. Additionally the primary Internet software technologies (summarized in Table 1) that are responsible for that change have been discussed at length as well. Table 2 summarizes and compares the inter-application communications technologies that were discussed in the latter part of the chapter. These comparisons will be used in Chapter 3, which describes a methodology to apply these techniques to provide a web interface to existing databases.

Table 1. Web Technology Features

Function	Technologies	Source	Web Servers	Browser Platform	HTML I/O	Language	Processing
Client Side Scripting	JavaScript	Microsoft		Major[13]	Yes[14]	Java based	Client Side
	JScript	Netscape		Major[13]	Yes[14]	Java based	Client Side
	VBScript	Microsoft		Microsoft	Yes[14]	Visual Basic	Client Side
Downloaded Complied Components	Java	Sun		Major	No	Java	Client Side
	ActiveX	Microsoft		Major	No	Any language and tool that can produce an OCX file	Client Side
Server Side Processing	Java Servlet	Sun	All		Yes	Java	Server Side
	ActiveX Server Pages	Microsoft	Microsoft IIS		Yes	Any	Server Side
	CGI	Various	All		Yes	Any	Server Side

All = Should work on current versions of all known browsers or servers

Major = Should work on widely used browser versions to include at least but not limited to Internet Explorer and Communicator.

Table 2. Comparison of Inter-Application Communication Techniques[15]

Technology	Object Definition Language	Interface Definition Language	Connection Abstraction Level*	OS Platform Support*	Ease of Configuration*	Multi-lingual Object Invocations*	Scalability*	Security*
CORBA	Any	CORBA IDL	4	4	3	4	4	4
RMI	Java	Java	4	4	3	0	1	3
DCOM	Any	ODL	4	2	0	3	1	4
Sockets	Any	Custom	1	4	1	4	4	3

*Scale : None 0, Few 1, Some 2, Good 3, Excellent 4

[13] Some JScript functions only work in Microsoft Browsers and some JavaScript functions will not work in Microsoft Internet Explorer [Gessing97]

[14] Ability to manipulate web page on the client side (forms processing, etc)

[15] Based on table in [Orfali97] page 333

(this page intentionally blank)

3 Methodology for providing web-based database access

This chapter presents a methodology for determining which Internet technologies to use in providing web-based access to existing relational data sources. The goal is to assist the developer in determining the right technologies, or mix of technologies, to accomplish the goals of their database access project.

By focusing on three essential factors affecting the design process and the capabilities of each Internet technology, a framework in which to narrow the large number of technology choices to a more manageable size is provided. As a result, the developer can then focus attention on the design rather than the nuances of the technologies themselves.

3.1 Methodology steps.

This methodology to aid the designer of a web-based access project consists of several steps. Each step and sub-step will be discussed in the sections as indicated in ( ).

- Development Environment Analysis (3.2)
  - Client Environment (3.2.1)
  - Existing Data Source Architecture (3.2.2)
  - Existing Resource Environment (3.2.3)

- Component Analysis and Design (3.3)
  - Functional Analysis Overview (3.3.1)
  - Component Functional Decomposition (3.3.2)

- Function Implementation (3.4)
  - Component Interface Definition (3.4.1)
  - Implementing Sub-components (3.4.2)

3.2 Development Environment Analysis

The first step in the methodology is to a analyze how certain aspects of the overall system environment can impact or influence the technologies chosen for implementing data access. The environment to be studied consists of the clients who will access the data sources, the existing database architecture, and the availability of development and maintenance resources (Figure 11).

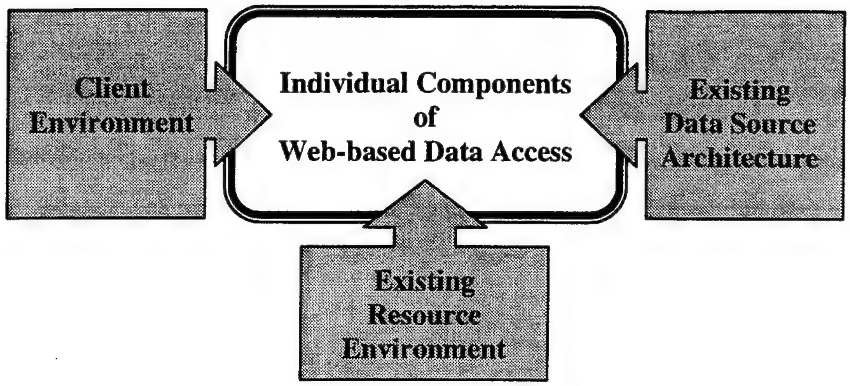

Figure 11. Factors influencing component development

With these in mind, the overall project is defined as a group of individual components, each providing a unique function. Each component, comprising the overall project, can then be modeled, designed, and implemented individually.

3.2.1 Client Environment

When implementing web-based access to databases, perhaps the most critical aspect of the design process is a thorough analysis of the clients that will interface with any components created.

There are two classes of clients in a typical web-based access scheme. The first consists of *internal clients* who are members of the organization owning the data sources. The second, *external clients*, are those clients who exist outside the organization, but who have a need to access the data. The major difference between the two will be in which functions (components) of the system are accessible to them and their network connectivity (Figure 12).

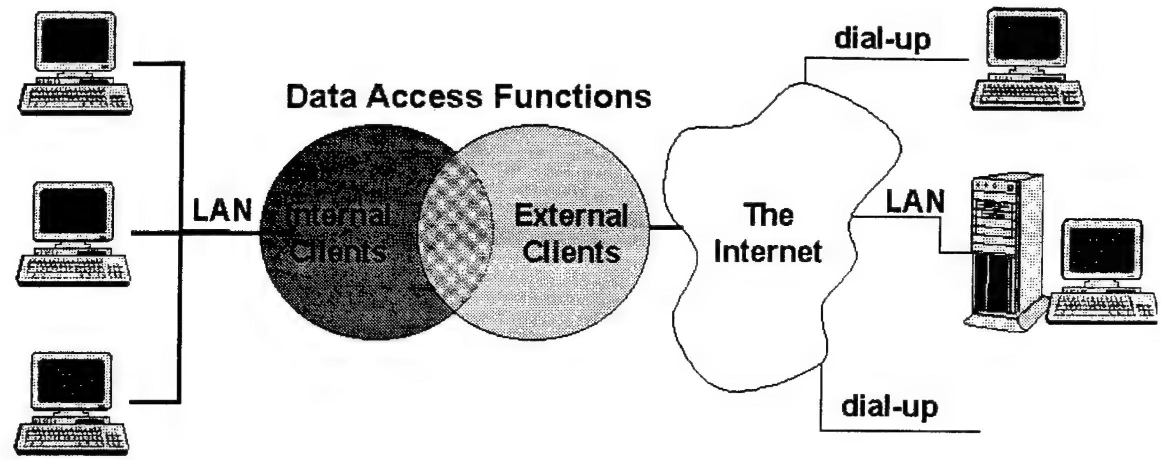

Figure 12. Client Environment

3.2.1.1 Internal clients.

In general, a developer will know the number of potential internal clients and the hardware and software platforms they use. He can therefore exercise greater flexibility in choosing implementation technologies that favor performance and integration over portability. As an example, consider a situation where internal clients use only the Microsoft Internet Explorer browser in the Windows 95 operating system and the server runs the Microsoft BackOffice server suite. The designer may choose to implement components for internal access using ActiveX controls that are designed to support Microsoft platforms. This will create a component that is tightly coupled with not only the server side of the system, but the client side also. The benefit of this tight coupling is increased component capability and better run-time performance since the component and client platforms were designed to work together efficiently.

Another positive aspect of the internal clients is that they are most likely to have a high-speed connection to the server, since they will commonly be on the same network or LAN. The developer need not be as concerned with the communications download speed of these clients as with external clients. More powerful components such as ActiveX controls and Java applets, that due to their potential size might not be desirable to use for external clients, can therefore be used

with greater frequency.

3.2.1.2 External clients

While much is known about the internal clients, the opposite can be said about external clients. Occasionally, the developer may know the number of external users who will interface with a function if it is designed for a number of approved clients. However, if the goal of the function is to achieve the widest access to the data resource then the number of potential clients can be unlimited and therefore much harder to predict.

The developer should plan for the worst case, that the latter is true, and he can only estimate the potential size of the external client base. The number of external clients can have a great impact on a web server's performance. This is because as the number of concurrently connected clients increases the server's processing load and can cause it to get overburdened if it is not of sufficient processing power. Also since a server typically has only one network connection, which must be shared by the data of all the clients, response times can be affected by a communications bandwidth bottleneck. The developer must take care to weigh the potential impact of an increased client load on the scalability of the system by assessing its effect on network bandwidth and server performance.

Two other areas of concern when developing functions for external clients is the software and hardware platform on which they operate. Since there are web browsers for almost every major operating system and hardware combination, and the number of clients using these platforms is largely unknown, the designer must always consider portability when designing components to be accessed externally. The impact of the choice to use a proprietary technology could cause unpredictable results. Some clients may experience no degradation in component functionality, others may get limited functionality, and some can have a total loss of functionality [Meuller97]. For example, consider an applet created in the Java programming language version 1.1.x. A client using Sun's HotJava browser will see the component exactly as the designer had

intended. A Microsoft Internet Explorer client may see the component, but GUI buttons or input form tabbing may not work correctly. In the worst case, the user won't see the component at all and therefore will not be able to use it.

Additionally, some functions that are supported on different browsers appear differently on each vendor's version, affecting the user interface to that function. For example, if JavaScript is used within a web page to show a help message for controls as the mouse passes over the control, it may appear over the item in some browsers and in the status bar in others. Figure 13 depicts a web page in which the designer has imbedded JavaScript code. It is evident that some unknown element of the code was unsupported in the browser, causing an error message to appear for the client.

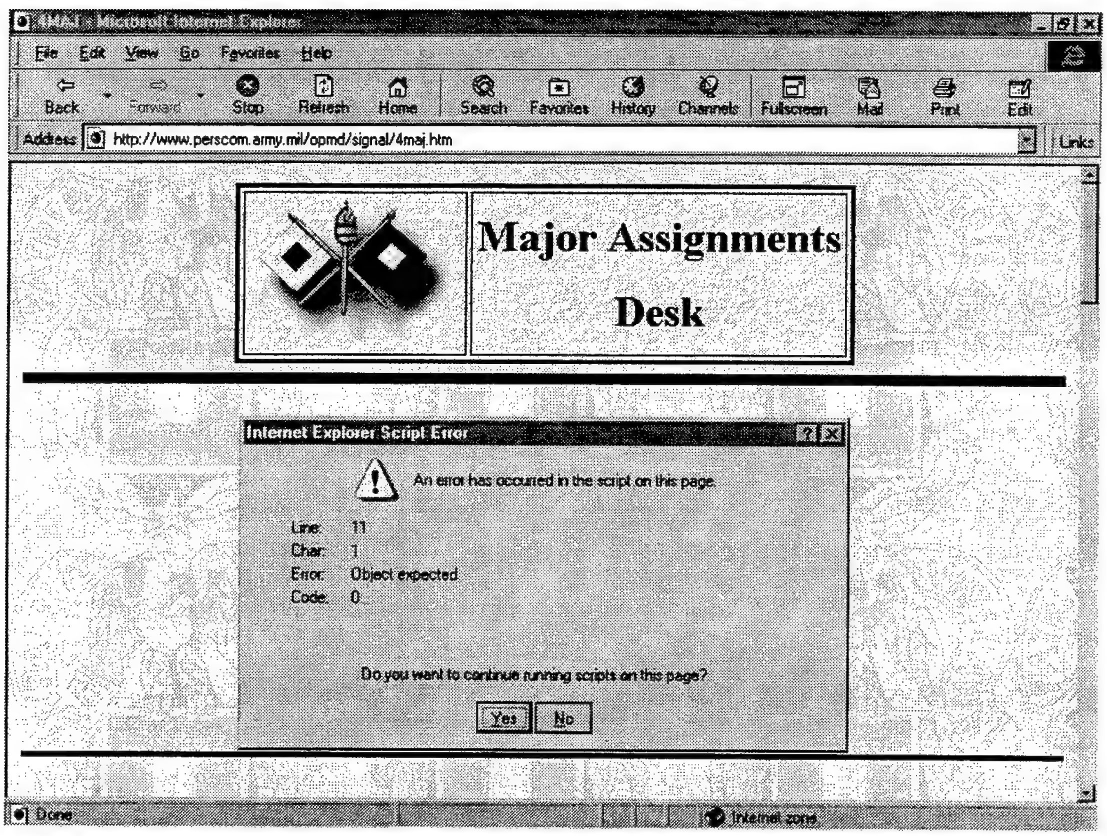

Figure 13. JavaScript Error in Client Browser

Another example of browser incompatibility is the way browsers display color. A developer may create and test a component on a machine that can display in a high resolution (SVGA) with 16 million colors, while a client is reading the page on a machine that is configured to support a lower resolution (VGA). The client's browser will pick the closest available color in its palette, which may not appear to the client as intended by the page developer.

Developers must also consider the external client's network connectivity when designing components to the system. The external client's connectivity could range from a direct T1 connection of 1.54 MBPS to a dial-up modem connection of 9.6 KBPS. Consideration must be taken for the client's connectivity in designing each component that will be available externally. For example, a developer implementing a function with a web page containing an applet, must take care not to make the applet so big that it results in a perceptibly unreasonable download time [Morrison97]. The performance loss due to long downloads is reduced, in some cases, if the web browser allows the downloaded components to be stored on the client's computer. Future execution of a stored component will use the local copy (if it is the most recent) instead of downloading it again. Presently, Java applets are downloaded with every access to the page and there is no persistence of the code on the client machine[16].

When designing a component, a reasonable analysis should be made of its download performance. Setting up a computer with a modem to estimate the worst-case access speed for external clients is one way to do this. Because a communications bottleneck can occur anywhere in the client-server link, the developer must also consider the server's bandwidth to the Internet when deciding on a technology for implementing functions for external clients.

To summarize, complete information about external clients is unknown and cannot be known with any certainty. Therefore, the components that they interface with should be designed for portability and of a sufficient size to optimize download performance. While the developer can not always plan for every external client situation, he should strive for a design that will work

for the most critical or largest possible set of external clients.

3.2.1.3 Overall Impact of the Client Environment

Clearly both the internal and external client platform characteristics will have an enormous impact on the design process. Since the whole purpose of the process is to service clients, this is understandably the most critical of the three environment factors discussed in this section. As long as software developers continue to develop browsers that support proprietary standards, the incompatibilities discussed will continue to exist. Because of this developers and systems designers must constantly consider the state of the client environment during the design phase of any web-based data access system to ensure that the requirements of the client are met.

3.2.2 Analysis of the Existing Data Source Architecture

The second area to be considered when picking technologies for web-based database access component designs is the existing database architecture itself. This includes any existing database management systems, their ability to handle an increased client load, and the physical location of the data sources.

3.2.2.1 Existing Database Management Systems.

An important aspect to consider when developing web-based database access components is the current architecture of the database(s) that will service the web clients. Two key areas of the existing DBMSs are of primary focus. The first is the multi-user capability of the existing database management software. The second is the existence of any original equipment manufacturer (OEM) or third party development tools (e.g., Oracle Designer 2000) for any DBMS that will be accessed. Each of these areas are discussed in detail below.

The ability of a DBMS to service remote web clients will depend greatly on whether it implements a single or multi-user environment. Since web clients accessing a database through

---

[16] Most browsers will load a recently accessed page from cache to save time when possible.

most interfaces are treated the same as local clients by most DBMSs, the developer can predict how the existing DBMS software will respond to an increased client load. For example, adding a sizable amount of web clients to a DBMS that was designed to be single user, or that can only handle a small number of concurrent users, will adversely affect both local and web user's access and performance. Additionally, many single-user DBMS's consider the database to be in use (locked) while a web connection is in progress. This can, at best, limit additional clients to read-only access, and at worst deny service altogether. Other DBMS's will exhibit severe performance degradation when accessed simultaneously by a large number of clients. For example, when connecting through an ODBC interface to a Microsoft Access database, the ODBC software actually opens the database as if it were being loaded locally from the full MS Access system. This will at best limit other users to read-only concurrent access of the same data source.

With the connection capabilities of the database in mind, the developer should not create an access demand that the DBMS was not designed to handle. If that is not possible the developer should upgrade the DBMS to a system that can serve not only the local users of the data, but a large volume of web clients also.

Another area that will influence any design is the possible existence of OEM or third party development tools. The more prevalent and popular a DBMS is, the more likely it is that such tools exist. Many current relational DBMSs offer integrated web front-end development tools. These tools allow the rapid graphical design of web forms, reports and ad-hoc queries. These tools can also automatically generate the code necessary to compile these applications. This generated code is often in the form of a Java applet or other client-side component technology.

The major benefit of these development tools is their tight coupling with the target data source. The ease of use gives the developer the ability to rapidly create components to access their database. The tight coupling between the code created by the tool and the DBMS ensures that the web-components will likely be more functional, and have fewer bugs. Many of these

tools are also self-documenting, greatly enhancing maintainability of the code.

A negative aspect of these tools, however, is that they can often be prohibitively expensive if not bundled with the database system. Some tools also require the use of software plug-in code, installed in the client's browser, to access the database. This further complicates the issue of client portability in two ways. First, it introduces not only the requirement to have versions of the plug-in for several browsers, but also potential difficulties that can occur when the client attempts to install the plug-in in their browser. Secondly, others may implement front-end functions in a technology that is not supported in all web browser versions creating incompatibility between clients and the server.

The third issue is the available methods of access. Some databases can only be accessed via the web through a proprietary interface. This will limit a design to the methods supported by that database management system, while other DBMS systems have multiple access methods that make them an easier and more appealing option. Many DBMS systems such as Microsoft Access can communicate with clients through a number of means such as a JDBC-ODBC [JavaSoft97a] bridge or through Microsoft's DCOM [Mueller97]. Many database vendors are developing JDBC and ODBC drivers for their database systems also [JavaSoft97a]. In general, the more methods of connection available to the developer, the more flexibility he has in choosing a technology to use in implementing a specific component.

After the client environment, the existing database architecture is the next most important factor to consider when designing web-based database access components. The developer must be aware of the capabilities of the DBMS and how it will scale-up with additional clients in order to keep from overloading the DBMS with additional connections.

3.2.2.2 Location of Data Sources

The second area of the existing DBMS architecture that deserves consideration is the physical location of the data sources. There are two general cases, the first of which is that the

desired data resides in a single database or in multiple databases on a single server. The second is when the data to be accessed resides on different machines.

The second case will be the primary concern. The fact that a web component needs to access data on several different machines can affect a design by increasing its complexity and introducing potential security and performance issues.

When designing a component that provides access to a single database, the task is simplified by only having to interface with one data source. The addition of a second source, however, can mean that the developer must write code to deal with two servers who may provide access to their data in different ways. As an example: an application component needs to access database A on one machine which provides access through a JDBC-ODBC driver, and data on machine B that is only available through a CORBA interface. Assuming the developer wishes to use an applet that is sent to the client with a web page, he will have to include the necessary code to access each separate database within the applet. This adds to the size and number of class files that will have to be downloaded to run the applet, making the code more difficult to create and maintain.

Attempting to access several machines can also violate a browser's security constraints. This is because, as a security measure, most web browsers will only let applets connect to the machine from which they were served the web page. Although the latest browsers are allowing this access through several different means[17], the designer still risks denying access to a large number of clients that use an older browser.

The additional code complexity and security protocols involved in accessing several databases on different machines also decreases the performance of the client by introducing more processing and communications overhead. Accessing the data directly from each database, however, is not the only possible solution to be considered. Others solutions include using a

[17] This can be done through the use of security certificates or the voluntary relaxation of the browser's security parameters.

three-tiered approach as discussed in chapter 2, (section 2.3.2) in which a server process on one machine handles the interaction with the databases by receiving requests from the client and executing them on the appropriate database servers. The aggregate results are then passed back to the client from this middle-tier process.

The benefit of a three-tiered approach is that since the middle-tier is an application rather than an applet it can connect to other machines without the security restrictions imposed by the client's browser. It can also maximize the performance of data access by residing on a server with a fast communications path to the database. In a three-tiered approach, an applet can communicate with the middle-tier using a variety of interfaces such as TCP/IP sockets, RMI, CORBA, or DCOM[18], or even through custom protocols.

It has been shown how the method of access chosen by the developer has a significant impact on performance. Therefore, to ensure that any performance loss is kept to a minimum, the developer needs to pick access and techniques such as using a three tiered architecture to help in this regard.

3.2.3 Existing Resource Environment

The third factor to consider when analyzing how to implement web-based data access is the existence of available resources to aid in the design process. These resources consist of time, money, and personnel. The availability of these will not only affect which tools and techniques a developer can use to design components, but also how much effort he can expend analyzing which of the many choices can be used to solve his data access problem. The methodology described in this chapter will aid the designer who is limited in at least one area, but most likely in two.

---

[18] As discussed, these standards may or may not be supported in a particular browser platform. For instance Microsoft has sworn to never support Sun's RMI because they believe DCOM is better.

3.2.3.1 Available time for development and maintenance

Time, and its availability, can not only limit how the designer develops components, but also how well they can be maintained. A designer who also has to perform many other duties in the course of his work may lack the appropriate development or maintenance time needed for large-scale original system development. He should therefore seek to build components that are simple, clearly and completely documented, and that maximize code reuse whenever possible. If time is a limited resource, he may not have the ability to become familiar in detail with every possible technology before building any components. Because of this, the designer should at least be familiar with the capabilities and drawbacks of each technology in order to help him decide which best suits his design goals and environment constraints.

The ability to maintain components may also be affected by the availability of time. By maximizing code reuse, writing clear and complete documentation and using accepted and well-supported technologies, the designer can simplify any necessary code maintenance. If he uses a technique that is not very well supported in the computer industry, or creates a 'stove-pipe' system, he may create a component that while perhaps small enough to develop in a time-limited environment, is very difficult and time consuming to maintain. For example, there is a wealth of knowledge on Java and the number of books being written on it is steadily increasing, so a developer can easily find answers to questions or help on a problem.

Reusing existing, well-documented code when possible can ease many documentation difficulties due to time constraints. Another item in favor of the developer in this regard is that some development tools such as Java are self-documenting or can automate the process of documentation.

3.2.3.2 Available funds for development

Another resource that is often limited during development of components is funds. Budgetary constraints may provide a discriminator that eliminates one or more of the available

design tools. Although hardware upgrades may be necessary to support the increase in clients to a system, the developer should focus on the impact of software costs since it can be the biggest discriminator in choosing technologies for any implementation. For most cases, the cost of the client-side software, i.e., the browser, will not be a large concern since it usually already exists on the client's machine. Therefore the two items a developer may have to seek resources for, which are compared here, are the web server software and the development tools used to build components. Table 3 shows current pricing for some prevalent web servers and the operating systems they support. The level to which certain software technologies are supported can also vary between web servers, so designers should be familiar with the servers and the tools they support (Table 3).

Table 3. Sample of Web Server Pricing[19]

Web Server	Supported OS*	Server-side Processing**	Vendor	Price[20]
Apache Web Server	AM, U, O, W, WN	C, J	Apache Group	Free
IBM Internet Connection Server	AS, O, MV, U	C, J	IBM	$0 - $99
Java Web Server	O, W, WN, U	C, J	Sun Microsystems	$ 295
Jigsaw Web Server	M, O, W, WN, U	C, J	W^{3}Consortium	Free
Microsoft Internet Information Server	WN	A, C, J	Microsoft	$ 2179.95[21]
Microsoft Personal Web Server[22]	W, WN	A, C, J	Microsoft	Free
Netscape Enterprise Server	U, W, WN	J, C	Netscape Communications	$0 - $995
Oracle Web Server	U, W, WN	J, C	Oracle	$2495

* AM – Amiga AS – AS/400 M - Macintosh MV - MVS O - OS/2 W - Windows 95 WN – Win NT

** J - Java servlets C - CGI A - ActiveX Server Pages

Additional costs can also arise from the developers need to use development tools in the creation of applications. For example, if he wishes to use ActiveX components in his design he will need to obtain Microsoft Visual C++ or Microsoft Visual Basic in order to create them. Applets are written in Java, which is currently free. However, some integrated development

[19] There are other web servers, too numerous to list, ranging in price from $0 to $40,000 [Hoffman97]
[20] Current prices as of 18 Nov 97 from various sources
[21] IIS is an individual component of the Microsoft BackOffice server suite of software.
[22] This is a limited capability version of Microsoft Internet Information Server that will support a smaller number of web clients

environments that can be used to develop components can be quite expensive. Table 4 shows the current costs for a small sample of these tools.

Table 4. Sample of Development Tool Costs

Development Tool	Purpose	Vendor	Price[a]
Integrated Java Development Environments	Applet, Application Dev	Various	$50-$1000
Java Development Kit	Applet, Application Dev	Sun Microsystems	Free
Java Servlet Development Kit	Java Servlet Development	Sun Microsystems	Free
Microsoft InterDev	ASP Development	Microsoft	$ 215.00
Microsoft Visual Basic	ActiveX Development	Microsoft	$ 429.00
Microsoft Visual C++, Professional Ed.	ActiveX Development	Microsoft	$ 429.00
Microsoft Visual J++	Applet, Application Dev	Microsoft	$ 80.95

Overall, the impact of funding resources is that it can limit the tools at the developer's disposal. This, in turn, can determine which technologies are used in any implementation (Chapter 2, Table 1). Conversely, if the developer wishes to use a particular technology, such as ActiveX, in his components, this will possibly require the funds to purchase a tool that can produce ActiveX controls.

3.2.3.3 Availability of development personnel

The last development resource considered is personnel. The abundance or lack of personnel during the analysis and design phase will have great impact on how the developer chooses technologies for implementing database access components. If there are ample personnel for analysis and development, a more in-depth analysis can be done of the competing technologies. The additional personnel can also reduce the overall time needed for component development.

Different personnel can simultaneously evaluate different implementations of the same component to determine which creates the best component in terms of portability, performance, and functionality. Additionally the developer can delegate the design of components comprising

a database access project, thus reducing the time taken to analyze, design, and implement the overall project. However, if there is a shortage of development personnel, the available time for analysis and design will reduced. The developer may then choose to produce components using a technology that he is familiar with or one that develops a web component through a development tool included with their DBMS.

Since those two methods may not be the best for the project, the drawback of using this approach is that the speed at which the components can be developed may come at the expense of client-side portability and performance for both the client and server.

Overall, if the developer is limited in personnel, he should at a minimum seek to learn the capabilities, drawbacks, and communications capabilities of the technologies to be considered before starting the analysis and design process. This way he can at least assess the major tradeoffs of competing technologies.

3.3 Component Analysis and Design

The previous three sections have discussed how the client environment, existing database structure, and available resources can influence the design of individual components. It will be important for the designer to have a clear understanding of each of those areas as he develops any components to access data over the web.

This section takes a structured approach to creating the individual components that will make up an overall web-based access project. The methodology to be discussed will guide the developer in deciding which technologies will meet the specifications of each individual component.

A component, in this sense, is an application or combination of applications, that implement one clearly defined function. A component itself may require several sub-components to perform its desired function. To illustrate this point, an example component to reserve a library study room will be analyzed and designed. The end product will be briefly described

followed by an explanation of how that solution was derived. The example will involve five sub-components. The first sub-component obtains the desired time slot from the user. The second queries a database to find out which rooms are available in that period. The third and fourth display the results to the user and handle a room reservation request respectively. The fifth sub-component processes, then displays the results of the request back to the user. Because the sub-components are not required to be part of one big application, they may be implemented using different technologies. This gives the designer the flexibility to consider all available technology options that can implement the particular function of a component. This flexibility means that there can be several solutions to any component design problem. Each solution can be considered better than any others given the right set of design constraints. The constraints are those derived from the three environmental factors discussed in this chapter. At this point however, the designer should be concerned with the functional analysis of a component, saving the implementation technology choices as the last step.

3.3.1 Functional Analysis Overview

In performing a functional analysis of the component to be designed, an object-oriented (OO) functional modeling diagram technique called a Data Flow Diagram, or DFD, is useful to depict the component's overall function [Rumbaugh90]. In the DFD, processes[23] are represented as a circle containing a description of the process. Arcs leading into a circle denote an input to the process. Arcs leading out denote an output. Any persistent data stores[24] are represented by the database name with a line above and below. Arrows on the arcs leading in and out of the data source indicate whether it is read-only or can be written to. Figure 14 shows the top-level DFD for the example problem.

[23] Synonymous with "component" when used in the context of functional decomposition.

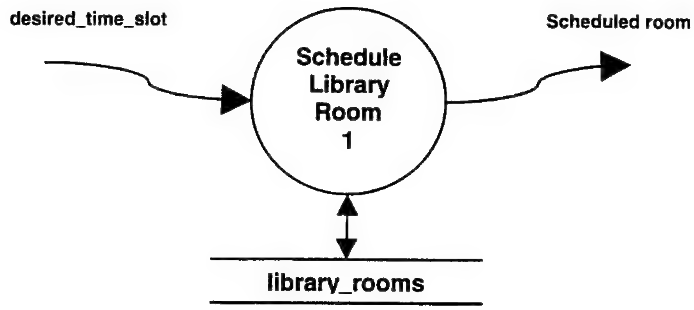

Figure 14. DFD for Schedule Library Room Component

Using functional decomposition, the component may now be decomposed into any necessary sub-components. As previously stated, the developer is not expressly prohibited from building a monolithic component that does everything. However, since different technologies can perform some functions better than others, that approach would not be desirable because it would limit him to one implementation technology.

The next section discusses the decomposition of the library room-scheduling component into sub-components. When this decomposition is complete, the methodology requires that the lowest level functions be analyzed to choose a technology to implement each lowest-level function. The level to which decomposition is performed is at the designer's discretion. It is important to note that although functional decomposition is used in this methodology to determine the sub-functions that need to be built, those sub-functions may be applications themselves. Therefore, proper software-engineering techniques should be applied recursively to analyze and design the individual sub-components in the technology chosen for them. In this case, the developer can implement the lowest level sub-functions (processes) in any technology. However, the individual sub-component itself will be built using a single technology such as a Java applet or ActiveX Server Page.

[24] These can consist of database files or other files used to hold intermediate results.

3.3.2 Component Functional Decomposition.

When decomposing the processes of a component into sub-processes[25] it is broken into smaller sub-components, whose function is a sub-task of the overall component process. In the fully decomposed DFD, the inputs and outputs to a sub-process may come from either the user or another sub-process. For example one component may take user input and send it to another component that queries the database and sends the results back to the first for display to the user[26].

Using the room reservation example, it is apparent that after decomposition, five sub-processes are required for implementation. As shown in Figure 15, the first function gets the desired start and stop time from the client. It then sends those times to the second function that must determine which rooms are available during that time slot. That process in turn sends the client the results of the query for viewing and reserving a room, if desired. If the client requests to reserve a room, the fourth process handles the request and sends the results to the fifth to be displayed for the client.

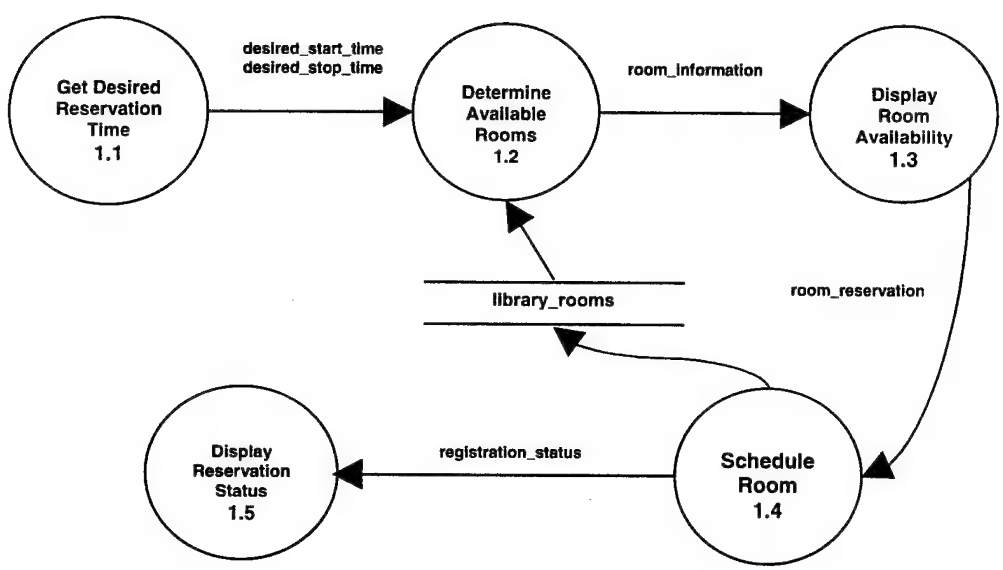

Figure 15. Lowest Level DFD of Room Scheduling Component

---

[25] A sub-process will be considered synonymous with a sub-component.
[26] This would be the case in a three-tiered component architecture

The *schedule room* component is now decomposed to the lowest desired practical level. In the implementation of the decomposition, the processes will be built using one of the Internet software technologies described previously. The arcs represent the interconnection of sub-components using one of the communications protocols discussed in Chapter 2. The data to be passed between sub-components is listed on the arcs.

3.4 Function Implementation

The next step in the methodology is to take the fully decomposed component (Figure 15) and analyze the possible technologies that cab be used to implement each sub-component process bubble. In the OO design paradigm, the developer would now implement each function as a method within an application program. While he may choose to build the component in this manner, he is not bound to do this. The developer may choose to implement any or all sub-components in any of the existing web technologies rather than just a single one.

Before picking technologies to implement each sub-function, it is important for the developer to be familiar with the methods in which the different technologies can communicate with each other. This is because the choice to implement a sub-component in a certain technology will limit what technology the developer can use for adjacent sub-components. For example a static HTML page can only originate from a web server process and can only send output to a sub-component that can receive an HTTP request. The next section looks at the interface protocols available to each technology.

3.4.1 Component Interface Definition

At this point it is important to look at each technology in terms of how it can interface with other technologies being considered. The following sections describe the interface of each technology and some implications of choosing that technology to implement a sub-process.

Downloaded compiled components (Java applet, ActiveX)

Compiled code that executes on the client-side can interface with other components with great flexibility because it supports almost all of the interface communications methods discussed in Chapter 2. This type of application is also capable of making calls to a database over the Internet (via JDBC or ODBC) provided that a network database service is running on the data server. Figure 16 shows the possible protocol interfaces to client-side code components.

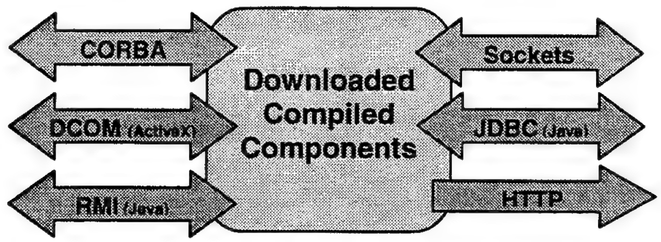

Figure 16. Interfaces to Client-side Compiled Components

Stand-alone applications

Stand-alone applications can interface with other components using both high-level protocols such as CORBA, RMI, and DCOM and low-level protocols such as sockets. This communications flexibility, coupled with their ability to perform complex processing tasks and database access (embedded SQL, JDBC, ODBC), makes stand-alone applications a powerful tool for applications in the middleware role of a three-tiered architecture.

Except for web server software (Apache, Microsoft IIS, Java Web Server), stand-alone applications cannot interface with other components through the HTTP protocol. Consequently, they can only communicate with downloaded compiled components on the client-side or web server processes and other stand-alone applications on the server-side. Figure 17 depicts the interfaces available to stand-alone applications. Note the only difference from Figure 16 is the absence of HTTP as a possible protocol.

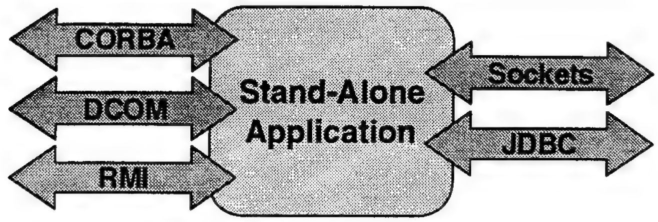

Figure 17. Interfaces to Stand-alone Applications

Web Server Processing (Java Servlet, CGI, ASP)

Web server processing components are very attractive because, unlike server-side stand-alone applications or client-side compiled code, they can receive input from and output to HTML documents using the HTTP protocol. This capability ensures that virtually all client browsers can use a function that is implemented with a server-side process. Since the web server process actually runs as part of the web server (a stand-alone application) it has the interface properties of stand-alone code. Consequently, it can make database calls and communicate to other processes using one of the supported communications protocols. It can also send HTTP requests to other server-side processes[27].

Figure 18 shows the interface to a web server process. Note the only difference between Figure 16 and Figure 18 is the addition of a two way arrow for HTTP indicating that web server processes can receive and process HTTP responses from other web-server processes.

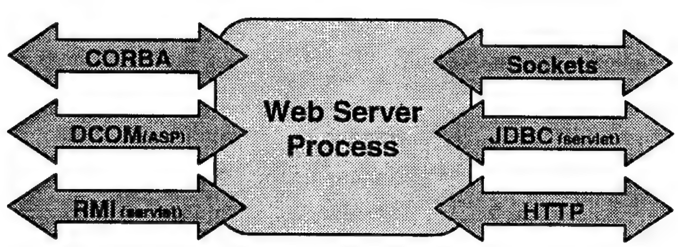

Figure 18. Interfaces to Web Server Processes

[27] Called servlet chaining [JavaSoft97c]

One limitation of interfacing with a web server process is that although it can easily communicate with stand-alone applications and invoke other server-side processes, it can only be invoked by an HTTP request. On the output side, the only interface it can use to communicate with the user is via HTTP as an HTML document. This means that any resulting output to the user will be in the form of a static web page. A server-side process can indirectly send input to compiled code such as a Java applet, but it requires the developer embed references to the compiled component in the web page returned to the client.

Static HTML page

A static HTML document is the simplest of components that can be used to implement a function. This section concerns HTML documents that do not contain any embedded client-side compiled code. HTML can be used quite effectively for data input and output. Graphical data input forms that can be viewed by all browsers can be easily created with only the standard HTML command set. In addition, the HTML code can be augmented with scripts that can do input data validation and other client-side tasks [Gesing97]. Figure 19 shows the interface to a static HTML document.

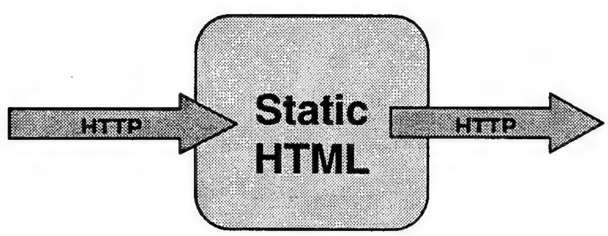

Figure 19. Interface to a Static HTML document

The static HTML document's simplicity is its greatest limitation. It can only be generated from a web server or web server process sent to the client as an HTTP response, and can only be used to initiate HTTP requests as input to other components. This precludes the use of an HTTP request in communicating with all components except a web server process (ASP, CGI, servlet) or the web server itself.

3.4.2 Implementing sub-components

The component has now been functionally decomposed and the interface properties of the Internet technologies analyzed. The information gained in that process along with any design constraints imposed by the three environment factors can be used to begin making choices on technologies for implementing the component.

The developer is now faced with the question of which sub-component to implement with a particular technology first. The answer is that any sub-component in which he is constrained by one or more of the three environment factors can be a starting point. This will usually occur in either a user interface, or a database access sub-component. For the example, the following assumptions are made regarding the design specification of the room reservation component:

- the component will be accessed by external clients only
- the clients use a variety of hardware platforms running a similar variety of operating systems (Unix, Windows, Macintosh, VMS, etc.)
- some users may have low bandwidth connections (dial-ups)
- the client interface should look the same on all platforms
- the user should be able to print the reservation confirmation for their records
- no database architecture changes are necessary to handle additional clients

From the specification, it is apparent that due to the heterogeneous environment of the clients and the potential for low bandwidth connections, that the starting point would be at the user interface. In the example it was decided that all interface with the user (client) would be through static HTML pages. This maximizes the sub-component portability (visible on all browsers) and minimizes its bandwidth requirements (no downloaded code sent to client).

Therefore referring to Figure 15, sub-component 1.1 (*Get Desired Reservation Time*), 1.3 (*Display Room Availability*), and 1.5 (*Display Reservation Status*) will be HTML pages.

From the analysis of the software technologies and their interfaces, it is clear that an HTML page can only generate HTTP requests and only a web server process can generate an HTML page as output. Because of this, and the fact that all web pages will be served from a Java capable web server, a Java servlet was chosen to implement sub-components 1.2 (*Determine Available Rooms*) and 1.4 (*Schedule Room*). Since a servlet is capable of making JDBC calls to a data source, the choice of a servlet will perform the sub-component functions adequately.

The implementation strategy for the component is now complete and the developer may now proceed with the analysis, design and coding of the sub-components using the software techniques he has chosen.

3.5 Summary

It has been shown that there may never be enough information to support the choice of only one particular technology for providing web-based data access. However, the three environment factors and knowledge of the capabilities of each Internet technology should be enough for a developer to make a choice. In the next chapter, the methodology outlined here is applied to implement three functional components that provide interaction with a relational database. The functions designed will be in the context of the case study (AFIT/CI).

4 Implementation of Methodology for the Case Study

Chapter 2 provides a technical overview of Internet software techniques that can be used to enhance the client-server environment by providing web-based access to existing relational data sources. Chapter 3 provides a methodology to aid in the analysis and design of the software components that comprise a web-based data access project. This chapter presents an application of that methodology to a test case by describing the design and implementation of several web software components to provide access to existing AFIT/CI data sources. These components or applications perform functions that will benefit the CI staff, students and prospective students by providing new or enhanced access to AFIT/CI's students, institution, and base support data.

The first component provides access to existing data on the academic institutions participating in the CI program. Prospective students can obtain information about the institutions and the programs currently attended by Air Force students. This access was previously unavailable outside the AFIT/CI program office. The second component allows a current student to view and submit changes to their personal data in the CI database. The last component will allow a CI student to submit training report input through the web to their program manager. This is an enabling technology to facilitate automation of an otherwise manual and error-prone process.

4.1 Analysis of key development environment factors

This section examines the three environmental factors discussed in Chapter 3, Section 3.1, as they relate to the AFIT/CI test case. AFIT/CI will be analyzed by looking at the clients of any web database access, the existing CI database architecture, and the available development resources for building the web-based data access components.

4.1.1 Analysis of the AFIT/CI client environment

Chapters 3, Section 3.2, states that the potential clients of any web-based data access fall into two categories, internal clients and external clients. The following two sub-sections discuss the AFIT/CI clients in each category and the impact their composition may have on the applications to be developed.

4.1.1.1 Internal clients

Internal clients consist of those persons who are assigned to or work on the AFIT/CI staff. Chapter 3, Section 3.2.1, states that because the developer has knowledge of the hardware, software, and network capabilities of these clients, he can be less concerned with portability of applications to different client platforms. For instance, if the internal clients use the Windows95 operating system and the Internet Explorer web browser, he can use technologies, such as ActiveX, that can do more client-side functions (printing, file access) with greater efficiency than alternatives such as a Java applet. The developer can also use the knowledge of the connectivity of internal clients to determine how large and complex an application can be without affecting client performance.

Size of client pool

The internal client pool consists of approximately twenty-five program managers and other staff personnel within AFIT/CI. The small number of clients in this category ensure that any server-side processing applications the developer chooses to create for this group will not overload the web server.

Operating system, hardware and web browser platform

All internal client computers are IBM compatible PC-based computers. Each computer runs the Microsoft Windows95 operating system and has Microsoft Internet Explorer (version 3 or 4) as its web browser software. While the developer can still develop applications using very portable technologies such as Java, this platform homogeneity gives him the ability to create powerful client and server-side applications that only run in Microsoft browsers and the Windows95 OS. For example, if he creates an application as an ActiveX component, it will only run on a Microsoft platform but will run at native speed and with native privileges (file access, printing, etc.), on the client's machine. Consequently, service to intranet (internal) clients will be very efficient.

Network connectivity

All internal clients are connected to the web server over a shared 10MBps LAN. This fast connection ensures that applications created for internal clients will not suffer any significant performance loss due to network traffic. Additionally because of the internal client's excellent network connectivity, applications developed as downloaded compiled code (applets, ActiveX) can be of greater size than those created for external clients who may have a slower connection.

4.1.1.2 External clients

There are two types of external clients interacting with AFIT/CI. The first type consists of students currently enrolled in an AFIT/CI sponsored program. The second is prospective CI students interested in obtaining information about the CI program, prospective institutions, and the application process. As stated in chapter 3, section 3.2.2, precise information on external clients can not be obtained with confidence. Because of this uncertainty, the developer must make an approximation of the number of external clients and an estimate of their possible platforms in order to design components for external client use. These estimates can aid the

developer in determining the necessary level of component portability and to ensure that any components designed for external use will not over burden the server.

Size of client pool

There are approximately six thousand long and short-term students currently enrolled in Air Force sponsored CI programs. Additionally AFIT/CI receives approximately 1000 applications for the various CI programs yearly from prospective students. Therefore the estimated upper bound for the number of connecting external clients can be estimated at around 7000.

Because most components are designed exclusively for either current or prospective students and all external clients are not likely to access the server simultaneously, the number of actual concurrent connections is typically a small percentage of that upper bound. Even with a smaller number of connected clients, the external client pool is sufficiently large to require that any components be designed to minimize the work of the server when possible. This will help to increase client performance. Using connection statistics maintained by most web servers, the developer could assess the performance and effect on the server of any external components. The assessment can then be used to perform any necessary optimization on the components or the web server during the maintenance phase.

Operating system and hardware platform

External clients are typically Air Force officers, enlisted personnel or civilian government employees who may access components from work, school or home. Clients who access components from work or school use many operating systems to include Unix, VMS, Windows and Macintosh. Those that access from home predominately use a Windows/PC or Macintosh platform. The only requirement for an external client's OS/hardware platform is that it be capable of establishing a network connection and running web browser software.

Because of this uncertainty and the diversity of client platforms, components providing external access should be designed to maximize portability.

Network connectivity

External client connectivity to any CI data will typically be through the Internet and the bandwidth of any such connections will vary greatly. Some external clients may connect through a high speed LAN (megabits per second), while others connect through a slower dial-up Internet connection (kilobits per second).

Developing large components may result in reasonable performance for clients connected over a LAN while the dial-up users may experience undesirable response times. Again the developer must plan for the worst case and design any downloaded components to be as small as possible.

Client web browser software

Consistent with the lack of information regarding their OS and hardware platform, the external client's browser software is also unknown. Even though the browser will likely be drawn from a small pool, as discussed in Chapter 2, current browsers vary greatly in some key respects. Therefore it is assumed that external clients may be using older browser versions that do not support some of the sophisticated features included in current browsers such as support for *frames*[28] or scripts. For instance, older browser software may not be able to run client-side code such as JavaScript, applets or ActiveX controls. Therefore components for external clients should be designed with technologies that are supported in the largest possible subset of available browsers. For example, because all browsers support standard HTML, a component that presents information to the client as an HTML document is very portable. Additionally the developer can

---

[28] Divides the browser window into sections displaying different HTML documents simultaneously.

create a smart server, which can deliver client-tailored components to ensure portability.

4.1.2 Analysis of the existing database architecture

This section examines the effect of the existing AFIT/CI database architecture on the design and implementation of any web-database access components that might be developed. The existing database management system and location of the existing data sources will be analyzed with respect to any design or implementation limits they may place on the developer.

Existing database management system

AFIT/CI currently uses Microsoft Access as their database management system. The local database tables are populated by data that is imported by download from the Air Force Continuing Education System (ACES) database. Any changes to data elements that originate from ACES are not updated locally, but are submitted as changes to the organization that maintains the ACES database. The CI database will then reflect the changes on the next download. Data elements that exist only in the AFIT/CI database are updated locally by AFIT/CI personnel.

Location of existing data sources

AFIT/CI has no web server organic to its organization. They instead use the AFIT web server to service web clients. Because of this, a copy of the CI database is maintained on the AFIT web server to be used in web-based access. Because the data and server reside on the same platform, applications can access data with greater speed by eliminating the network communications overhead. However, the drawback to this approach is that both the DBMS and web server have to share resources (memory and processor) which can result in lower overall performance as the server's workload increases. Additionally any updates to the database made through the web interface will need to be replicated to the original database.

4.1.3 Analysis of development resources

This section covers the effects of the availability of development resources on the design and implementation of web-database access components. The effects of development personnel, funds, and time are all examined to show how they can effect the developer's design and implementation choices.

Personnel

There is currently one computer support person in AFIT/CI who is responsible for the database operations to include the importing of the ACES data. Likewise, one officer has the additional duty of maintaining the AFIT/CI web site, with the assistance of each PM in maintaining the pages for the program they manage.

Aside from the computer support person, any web development or database work must be done in addition to any other daily duties. This affects the development cycle by increasing the time required to develop components and the need for outside personnel resources (AFIT/SC) to assist in development.

Funding

Funds for web development within AFIT/CI are not currently programmed into the budget. If the developer chooses to develop ActiveX components, the purchase of Visual C++ or Visual Basic would be required. Because AFIT/SC is using Microsoft IIS as their web server, developing components that run as web server processes will not be affected by a lack of funds. This is because IIS uses ActiveX Server Pages that are written in VBScript or JScript and require no compilation. ASP applications can be written with any text editor and are compiled by the web server upon being accessed.

Additionally ASP pages can be developed and tested on any PC running the Microsoft

Personal Web Server (currently free). This means that no expensive hardware or server software is needed to develop and test applications. ASP components can be designed and tested locally and the finished product can be ported to the web server (IIS) with little or no changes.

Time

As stated above, the requirement for web development to take place in addition to normal duties will lengthen the time necessary to develop components. This can only be reduced by obtaining outside development resources or by the use of a *twenty-five hour day*[29].

4.2 Component analysis and design

This section presents an analysis and design of the three components to be developed for the test case. Any requirements for the component are briefly described and then the component is analyzed using the functional decomposition technique shown in chapter 3. Following the analysis and design, technologies are chosen to implement the component.

4.2.1 Component 1 (Institution / Program Information Search)

The goal of this component is to provide prospective CI students with information on the institutions participating in the CI program, their associated degree programs, and available academic disciplines. This information can aid the prospective student in choosing a school or program to apply for. Currently this data is present in the MS Access database and only available locally to AFIT/CI staff. Prospective students must receive this information by contacting a program manager. This information exchange typically occurs through standard email, postal mail, telephone conversation, or unguided web searches. Thus, in conjunction with the case study goals, this component will make more efficient use of existing data by providing it directly to the prospective students. This will reduce if not eliminate the time spent by program managers

[29] Currently used successfully in both theory and practice by the U.S. Army

on questions dealing with institution information and provide focus and structure to prospective student searches.

4.2.1.1 Requirements Specification

The students should be able to search for school information based by state, degree program, academic majors, or any combination of the three. The client must have the ability to print any returned results.

The client should receive as a response to their institution query, a list of schools and AF-sponsored academic programs sponsored by the school. A client should then have the ability to select for display additional information about the institution, the supporting base, and host ROTC detachment (where applicable).

4.2.1.2 Functional Decomposition

This component queries the CI database for institutions based on three possible input parameters obtained from the client. The three parameters are the client's desired location (state), degree offerings or academic program. The output will be institution information satisfying the client's query. Figure 20 shows the top level (level 0) DFD for the institution search component.

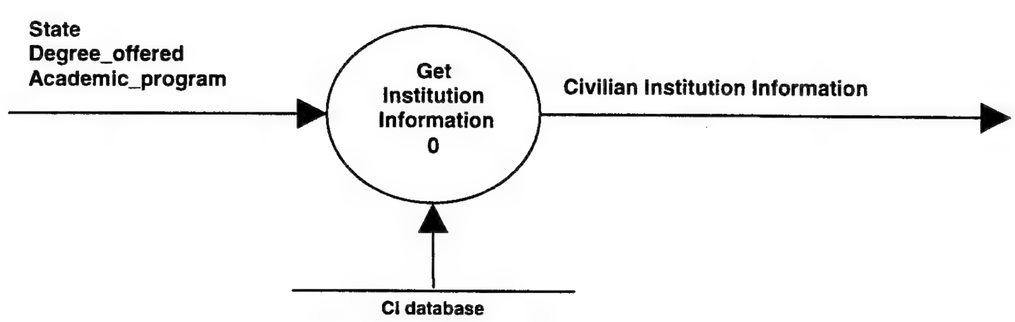

Figure 20. Level 0 DFD for Institution Search Component

The requirements specify that the student is able to obtain a list of schools, and then if desired, see additional information on a particular school. The component can therefore be further decomposed into the functions shown in Figure 21.

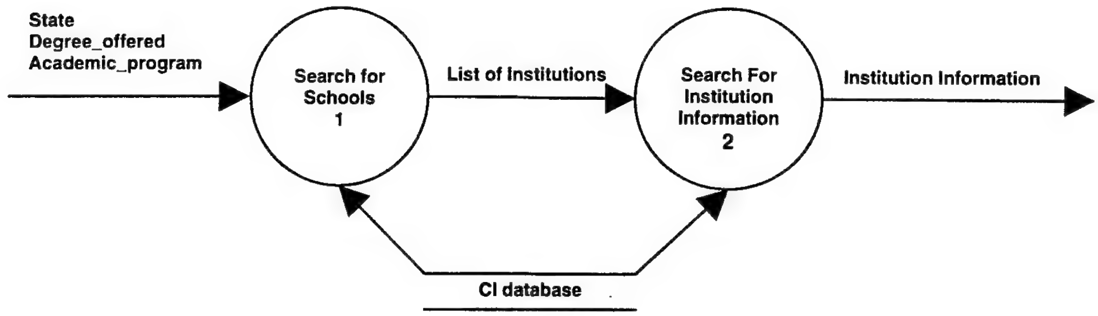

Figure 21. First Level Decomposition of Institution Search Component

In the next step the component is further decomposed to show the processes of the input from and display of data to the client (Figure 22). Because each circle now represents a clearly defined function that can be directly implemented in one of the software technologies discussed, no further decomposition is necessary. In the next section, specific technologies are chosen to implement each sub-component comprising the overall web component.

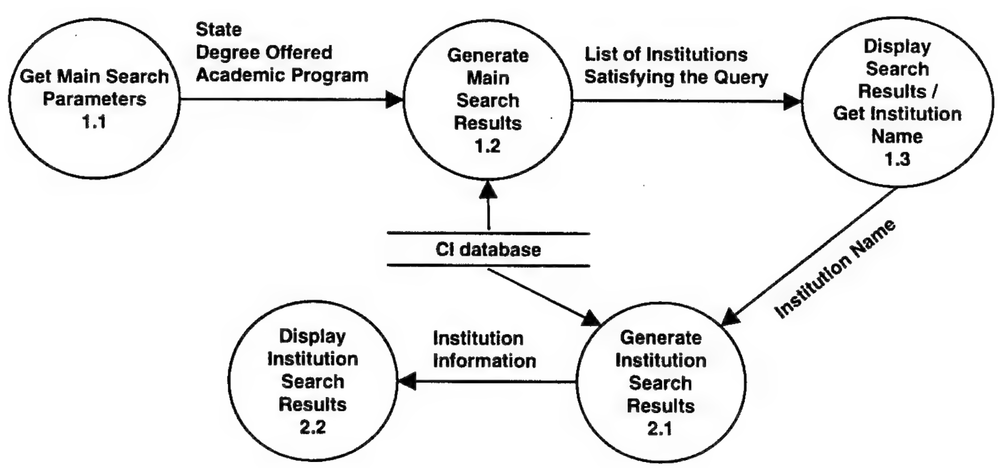

Figure 22. Lowest Level Decomposition of Institution Search Component

4.2.1.3 Design Implementation

As discussed in section 3.6 of Chapter 3, the designer must pick a sub-component as a starting point for choosing technologies for implementation. The requirement for clients to be able to print any results easily can be used as the basis for selecting the starting point in this scenario. This requirement means that any information displayed to the client is best displayed as an HTML document, because as described in Chapter 3, printing HTML documents is easily accomplished by the inherent printing capability of the client's web browser. Additionally, displaying data as HTML allows it to be displayed to the client in a more organized fashion using HTML's various text formatting and layout options.

Additionally static HTML documents can be viewed in all browsers. This satisfies the need to ensure that the greatest number of users can view externally accessed components. Therefore any client-side functions (process circle 1.1, 1.3 and 2.2) are implemented using static HTML pages generated by a web server process. In view of the interface capabilities of static HTML pages as discussed in section 3.5.3.1, only server-side web processes can return data in HTML through the HTTP protocol. The two server-side processes (1.2 and 2.1) are therefore implemented as a web server process.

Client-side sub-components

In addition to the requirement to print results, Process 1.1 requires the client to input data. To make the selection of query parameters simple, the client is shown a map of the United States and a data input form (Figure 23). The client can either click on a state to see all institutions and programs in that state or they may enter a more specific query by filling in the data input form. The data input form allows them to select institutions based on state, program, and course of study.

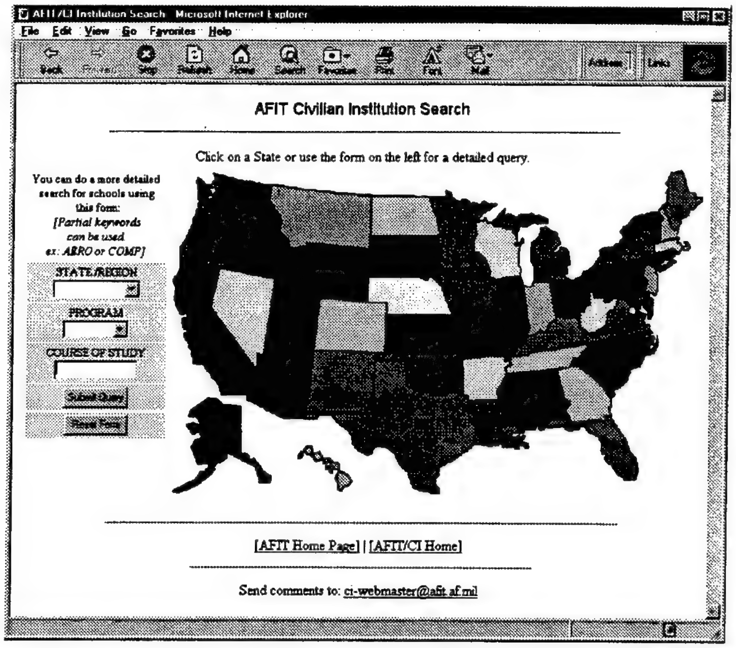

Figure 23. Screen-shot of sub-component 1.1

Sub-component 1.3 displays the results of the institution query (server-side process 1.2) to the client. Institution names are in the form of HTTP hyperlinks (Figure 24). Selecting an institution name calls process 2.1, which queries the database for information on that institution. The result of the query executed in process 2.2 is displayed as a static HTML page (Figure 25).

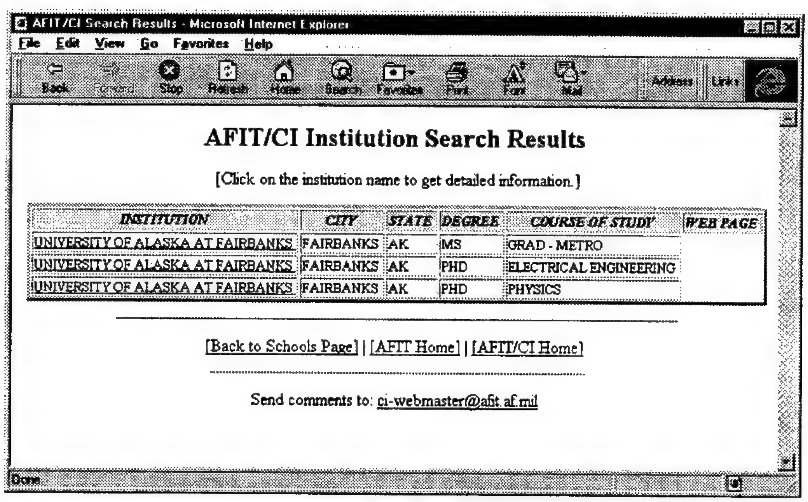

Figure 24. Screen-shot of sub-component 1.3

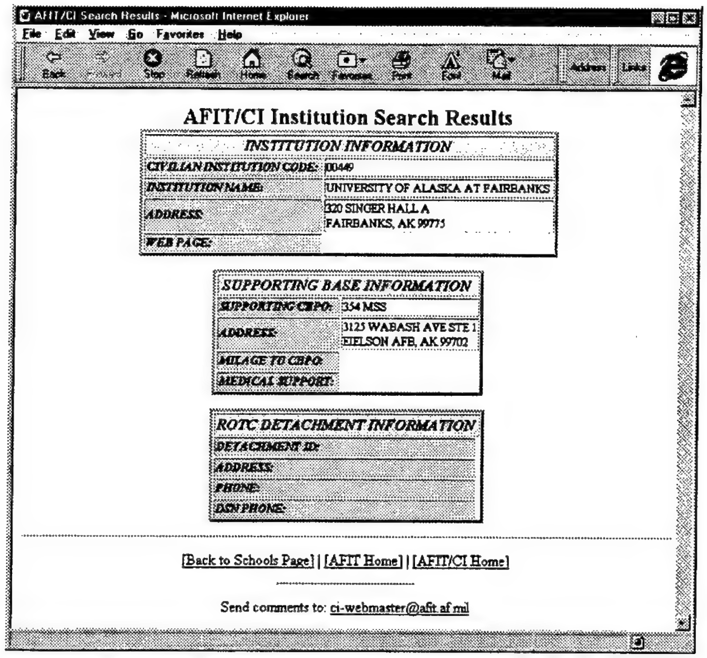

Figure 25. Screen-shot of sub-component 2.2

Server-side sub-components

As discussed in the analysis of the existing database architecture (Chapter3, Section 3.3) AFIT/CI will serve its web pages and applications from Microsoft IIS. Therefore, these sub-components (1.2 and 2.1) were implemented as ActiveX Server Pages, which can easily access data in the AFIT/CI database (MS Access DBMS).

The entire component (five sub-components) is implemented through the use of two ActiveX Server Pages. The first displays the image map and input form (Figure 23), processes the school search query, and returns the results to the client's browser as a static HTML page (Figure 24). If the client clicks the mouse on one of the school names, an HTTP request is sent to the second ASP which queries the database for information on a specific school and sends the results as a static HTML page to the client's browser (Figure 26).

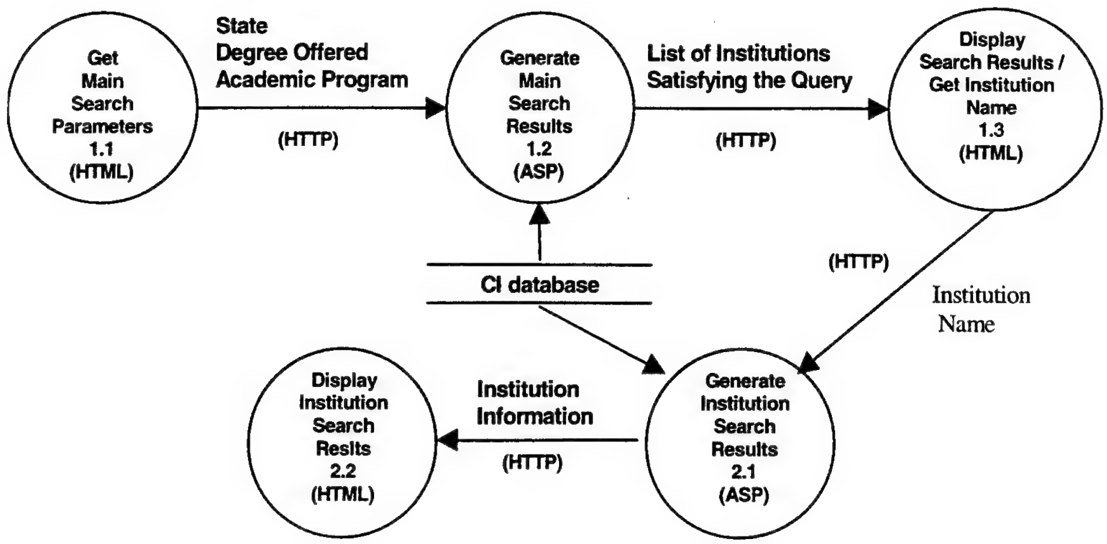

Figure 26. Component DFD with implementation technologies

4.2.2 Component 2 (CI Student Personal Data Update)

The goal of this component is to provide CI Students with a means to view their personal data as it exists in the CI database and to submit any changes if necessary. The changes are sent by email to the database administrator for posting. There was no previous means for students to see their information over the web, and any changes to their data were submitted over the telephone or on hard copy (fax or mail) to their program manager. This component therefore provides a previously non-existent service to the student and program manager.

4.2.2.1 Requirements Specification

Any students accessing this component must be authenticated to ensure data security and to aid in obtaining the correct personal data to display. The student must be able to view the existing data in a form with accompanying spaces in which they can enter changes. They should be able to mark any changed fields by placing an 'X' in a box next to the field to indicate that changes exist.

Upon submitting the changes, the application must email the CI database administrator with notification of the requested changes including student name, field name, the old value, and the new value. The student should get a response indicating if their changes were sent successfully.

4.2.2.2 Functional Decomposition

The component takes the student's last name and last four digits of their social security number as input (for authentication), allows them to make changes to their personal data, and outputs the changes in an email message to the CI database administrator. Figure 27 shows the top level (level 0) DFD for the component.

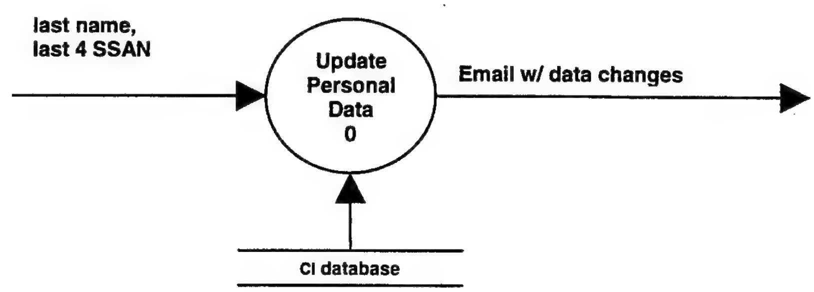

Figure 27. Level 0 DFD for Personal Data Update Component

The requirement for user authentication and email transport of any changes leads to the first level decomposition of the component displayed in Figure 28.

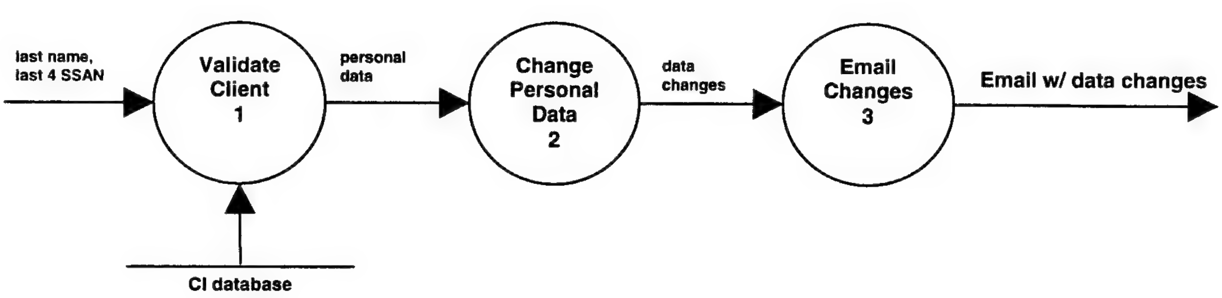

Figure 28. First Level Decomposition of Personal Data Update Component

In order to view the personal data and make changes, the component must query the database and present the data to the client in an HTML form. When the client submits changes, the application must email them to the database administrator and inform the client of the outcome of the submission (successfully sent or an appropriate error). These requirements lead to the further decomposition of the component shown in Figure 29. This is the lowest level decomposition of the component since each circle performs one unique function. In Section 4.2.2.3 further requirements and the existing development environment are analyzed to implement the sub-components shown in Figure 29.

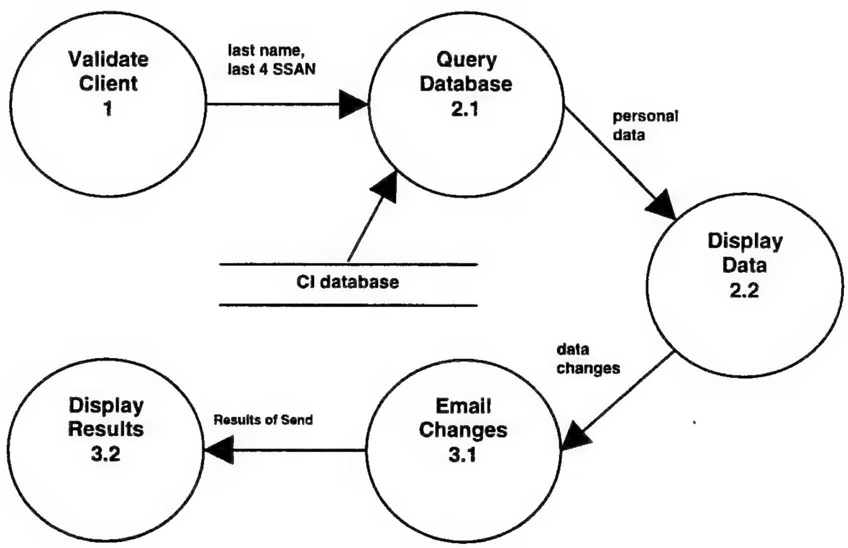

Figure 29. Lowest Level Decomposition of Personal Data Update Component

4.2.2.3 Design Implementation

This component will be accessed by external clients and therefore must be designed for client-side portability. The client's use a wide variety of platforms to access the data and therefore we will use standard static HTML pages for all the client interfaces (sub-component 1, 2.2, and 3.2). The requirement for client-side portability (HTML) is used as the basis for

determining which technologies we will use to implement the component.

The methodology from Chapter 3 allows that only server-side processes, specifically web-server processes, can generate static HTML pages sent to a client. We will therefore implement sub-components 2.1 and 3.1 as web server processes. From the client's point of view, sub-components 1, 2.2, and 3.2 will be static HTML pages. However, the web-server processes will generate the static HTML pages sent to the client.

Client-side sub-components

In order to meet the requirement for client authentication, sub-component 1 is a static HTML page (generated by a web-server process) in which the client can enter their last name and last four digits of their SSAN. This sub-component's function is implemented by a central application used to validate clients for all applications requiring authentication. The central authentication component (menu.asp) invokes sub-component 2.1 by with a valid last name and last four (SSAN) as input parameters. By pressing a button in the authentication application the client submits the data entered into the form to be authenticated. If they are valid clients, their data is sent to sub-component 2.1. Figure 30 shows a screen-shot of the sub-component 1.

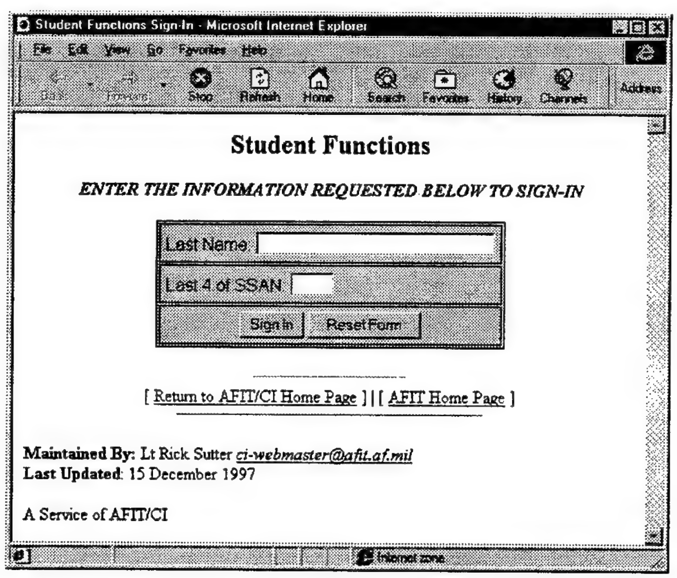

Figure 30. Screen shot of sub-component 1

Sub-component 2.1 queries the database for a record matching the last name and SSN (last four digits) supplied by the client in sub-component 1. If a match is found, a static HTML page is generated and sent to the client (sub-component 2.2). Sub-component 2.2 displays the personal data of the client. The client is allowed to indicate fields that need to be changed by placing an 'check' next to the field and by placing the new value next to the 'check'. Figure 31 shows the HTML form displayed to the client by sub-component 2.2.

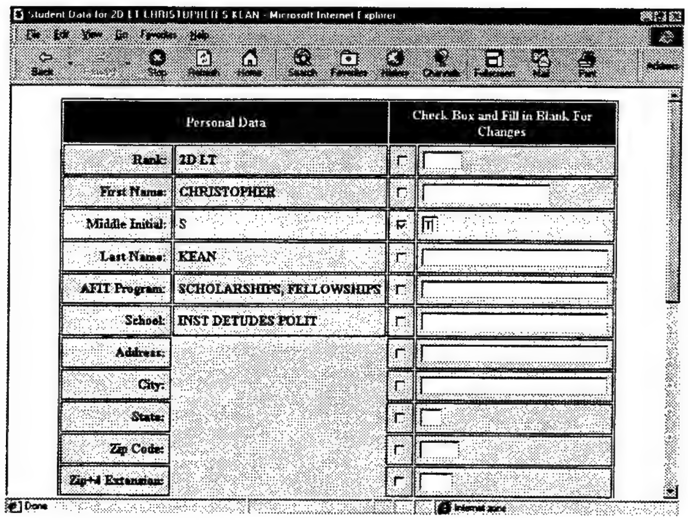

Figure 31. Screen-shot of sub-component 2.2

When the client has completed any necessary changes to their personal data, they press a button to submit the changes to sub-component 3.1 through an HTTP request. Sub-component 3.1 then builds a mail message containing only the data that needs to be changed. It then sends the message to the database administrator (Figure 32). Sub-component 3.1 (a web server process) then generates the static HTML page that informs the client of the outcome of their submission (Figure 33).

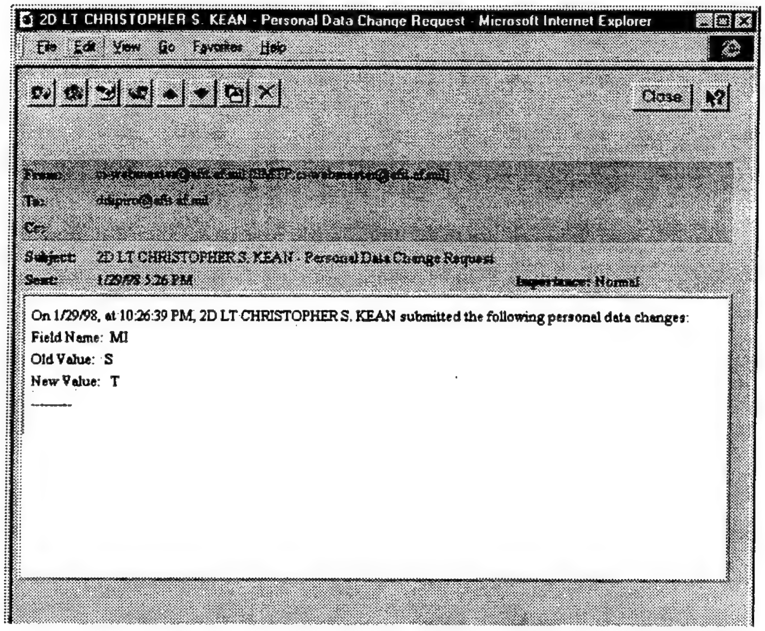

Figure 32. Mail sent to the database administrator

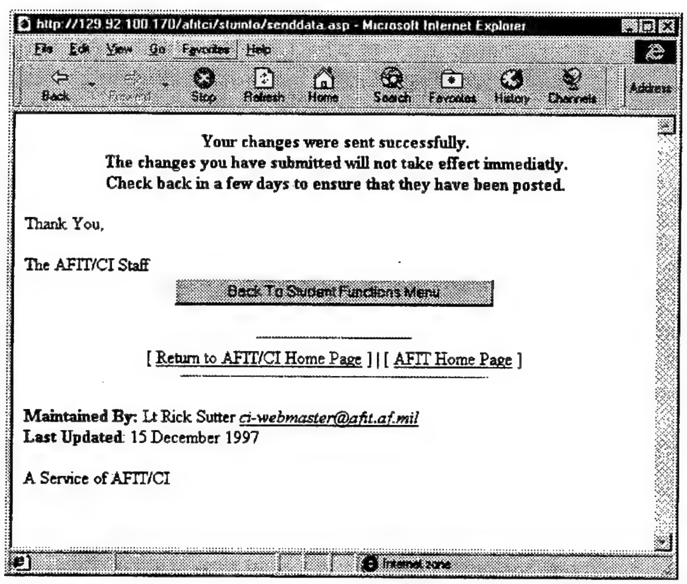

Figure 33. Screen-shot of sub-component 3.2

During the analysis of the existing database environment it was revealed that AFIT/CI would serve any data from a web server using Microsoft Internet Information Server. Sub-

component 2.2, and 3.1 will therefore be implemented as ActiveX Server Pages. Because ASP is the native web server processing method of IIS and is very capable of accessing the database, this technology can be used to accomplish the goals of the component. The client side static HTML sub-components, (1, 2.2, and 3.3) will be generated from ActiveX server pages (Figure 34).

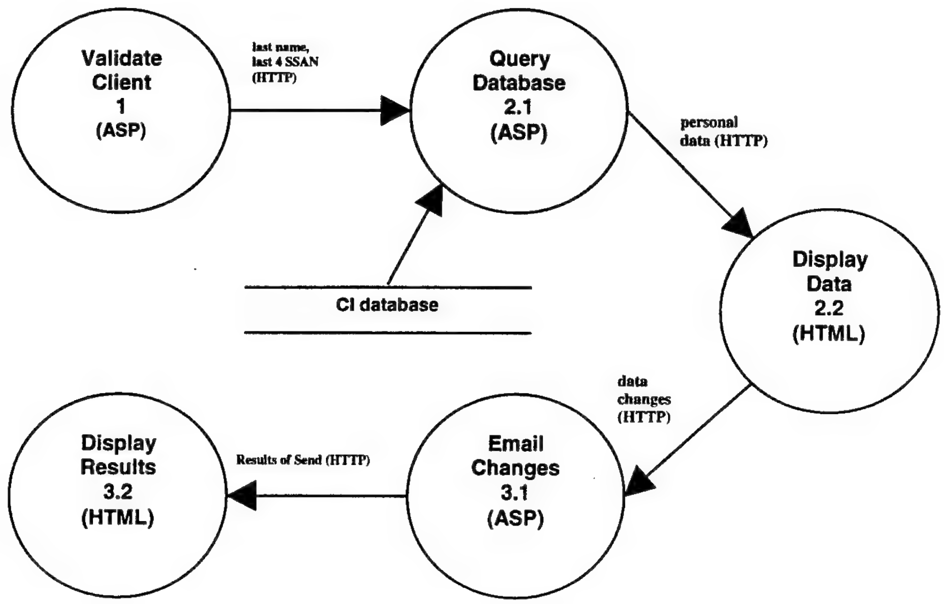

Figure 34. Component DFD with implementation technologies

4.2.3 Component 3 (CI Student Training Input)

The goal of this component is to provide CI students with an automated means to provide input to their program manager for use in the creation of their annual, directed, or final training report (TR), AF Form 475. This input allows the student to inform their program manager of any significant academic, professional, and personal accomplishments that can be incorporated into their TR. It allows the student to build a TR over the web and send it to their PM, who can read and modify it with an electronic form-publishing package (Delrina FormFlow) to produce the final document.

When building their TR over the web, this component builds a shell using existing data sources to fill in the student's administrative data and any previous TR comments submitted. The AF-475 shell is then sent as an email attachment to the PM as a FormFlow data file. The PM can then load the data file (email attachment) into FormFlow to complete the report.

Students currently send this input to their PM through email messages (text) or on hard copy (fax or postal mail). The PM must then enter the data into the AF-475 form, which is prepared using the Delrina FormFlow software. By using the existing database to build the shell, utilizing the web and email for transport, and by sending the shell in FormFlow format the process is quicker and less error prone. This component helps eliminate the errors that can come from manually building the report, which reduces the amount of time spent making corrections. This improves student-program manager communications and significantly reduces the overall TR creation process. It also gives the student significantly more ownership of the TR development process.

4.2.3.1 Requirements Specification

Any students accessing this component must be authenticated to ensure data security and to aid in obtaining the correct administrative data for the AF-475 shell. Authenticated students must be able to enter their input into a reasonable facsimile of the actual Form 475 to get an indication if they can include more or less input in the blocks of the form. The shell should be filled in automatically with the student's administrative data. If the student previously submitted TR comments, the component will also fill those in for reference. If the student enters comments for the first time, the application should store them for subsequent TR inputs. When submitting the completed shell the client should be able to select from a list of Program Managers who can receive the input. This eliminates the need to know who their PM is or their PM's email address. The data must be sent as email attachments in FormFlow format to allow the PM to easily load and edit the form.

4.2.3.2 Functional Decomposition

This component takes the student's last name and last four digits of their social security number as input (authentication), uses existing data sources to create and store an AF-475 shell, and sends the PM email with FormFlow data file attachments (the shell) as output. Figure 35 shows the top level (level 0) DFD for the training report input component.

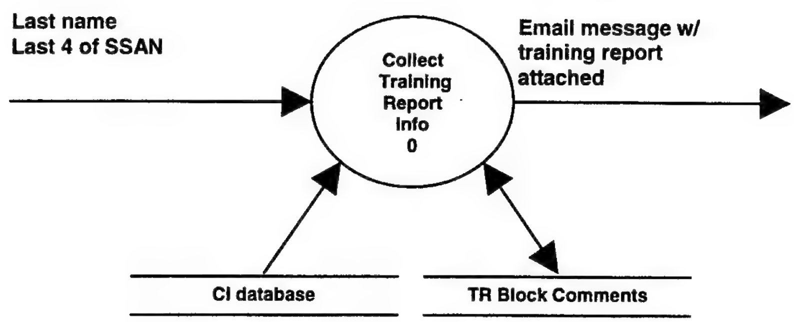

Figure 35. Level 0 DFD for Training Input Component

The requirements specify that the user must authenticate to use the component. It also stipulates that the resulting shell be sent (emailed) to the program manager. The component can therefore be further decomposed to the functions shown in Figure 36.

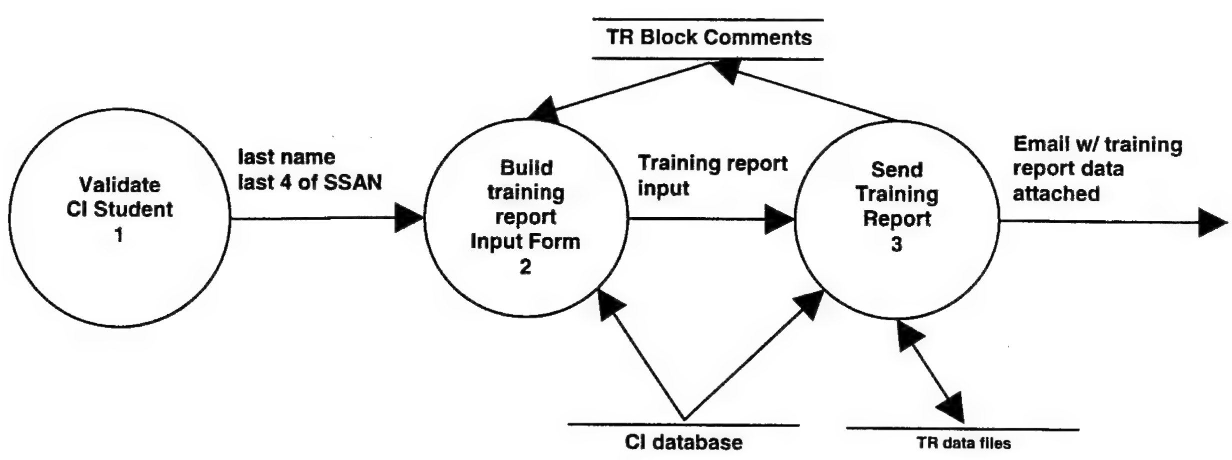

Figure 36. First Level Decomposition of Training Input Component

In order to authenticate the client and to get the appropriate administrative data for validated clients, the component must query the database on the student's last name and last four of their SSAN. If authenticated the component should display the AF-475 shell. To send the form as an email attachment, the component must also build the attachments (data files) from the shell data entered by the student. It must also save the comment blocks entered by the students. It must build a list of the program managers who can receive TR inputs so the student can choose a recipient. It should then send the form to the chosen PM and a response to the client indicating whether the mail was sent. Therefore, the component can be further decomposed to reflect these facts (Figure 37).

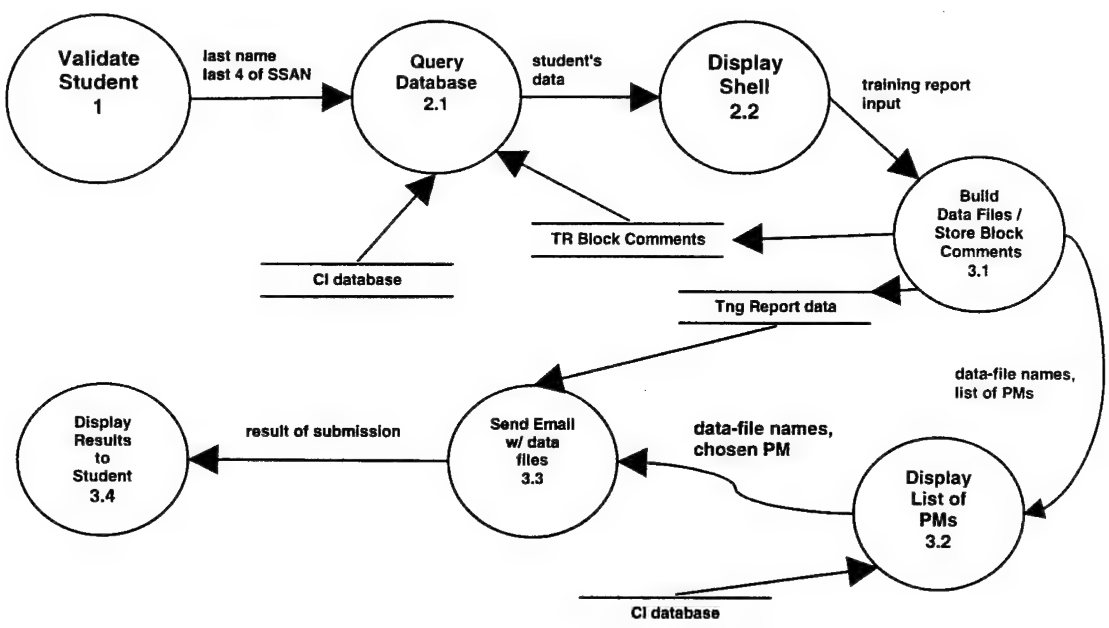

Figure 37. Lowest Level Decomposition of Training Input Component

Each process circle now accomplishes a distinct function that can be implemented using the technologies covered in Chapter 2. The next section shows the analysis of the requirements specification and existing environment factors in order to choose implementation technologies for

each of the sub-components (process circles).

4.2.3.3 Design Implementation

Like the first component developed (School Search) in Section 4.2.1, this component is to be accessed by external clients. The external clients using this component will be current CI students. Because it will be accessed by external clients using many platforms, the component must be as portable as possible. Therefore, all interfaces to the client should be in static HTML pages so that they can be viewed on the greatest possible number of platforms. We will use the need for portability (need for client-side static HTML) as the starting point when choosing technologies to implement the sub-components.

It is stated in Chapter 3, that only a web server or web server process can send static HTML to a client, therefore any back-end (server) processing (process circles 2.1, 3.1, and 3.3) will be accomplished through a web server process. This back-end processing will include any necessary database access, file I/O, and communications functions. All client interfaces (process circles 1, 2.2, 3.2, and 3.4) will be in the form of static HTML documents. Although sent to the client as static HTML, a web-server process will automatically generate these documents.

Client-side sub-components

In order to meet the requirement for client authentication, sub-component 1 is implemented by the same central authentication component as the component defined in Section 4.2.2 (Figure 30). This insures that any calls to this component come from a validated client. The input to component 2.1 will be that of a validated client.

Sub-component 2.1 queries the database on the last name and last four of the client. If a matching record is found, their administrative data and any prior TR input is obtained, and a static HTML page containing the AF-475 shell is displayed to the client by sub-component 2.2. Figure 38 shows a screen shot of sub-component 2.2.

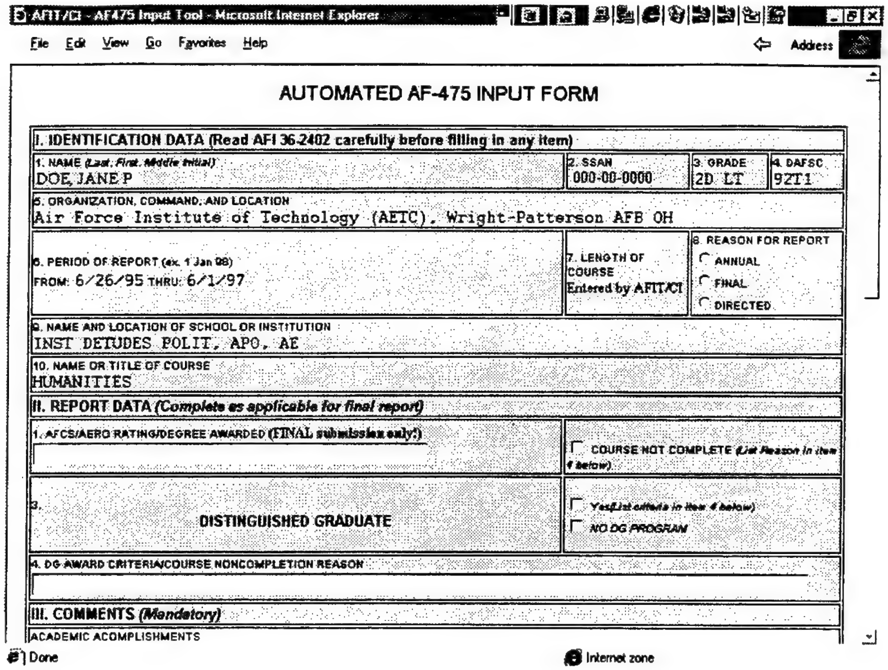

Figure 38. Screen-shot of sub-component 2.2

The client then enters their TR inputs directly into the form. When the client has completed their input, they press a button to submit the form for mailing to their program manager. Sub-component 3.1 then receives the form data and builds temporary data files (in FormFlow format) to hold the information. It also ensures that the block comments of the student are stored in a database for future TR inputs by the same student. The sub-component then queries the database to obtain the names and email addresses of the Program Managers and generates a page containing a list of available PMs which is displayed to the client in process 3.2. Figure 39 shows how the list of PMs is displayed, allowing the client to choose a PM to send the data files to. After the client chooses a recipient, sub-component 3.3 is invoked to send the data files to the appropriate Program Manager via email. The client will then receive a response (process 3.4) indicating the success or failure of their submission. Figure 40 shows the response for a successful submission.

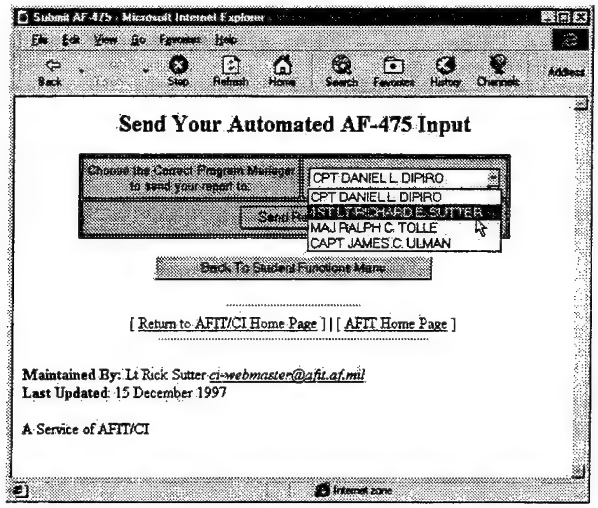

Figure 39. Screen-shot of sub-component 3.2

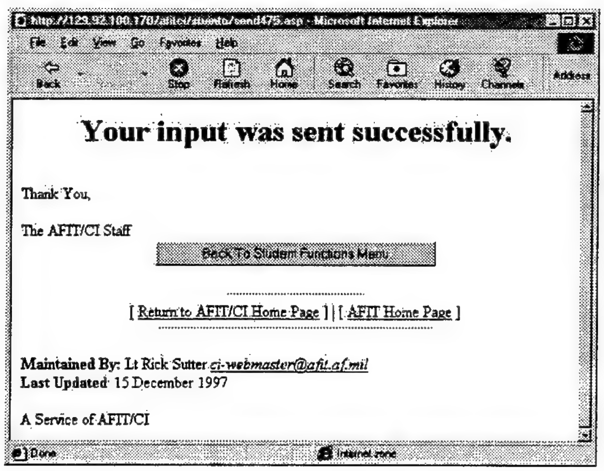

Figure 40. Screen-shot of sub-component 3.4

Server-side sub-components

Knowing that AFIT/CI will serve all web pages using Microsoft IIS, the sub-components implemented as web-server processes (process 1, 2.1, 3.1, 3.3) will be created as ActiveX Server Pages. ActiveX Server Pages will allow the file I/O, database access, and email functions that are required of this component.

The entire component (seven sub-components) is implemented through the use of three

ActiveX Server Pages. The first (af475.asp) receives the client's last name and last four and queries the AFIT/CI database to see if the student exists in their records. If a matching record is found, the client is sent a response containing a static HTML form that allows them to enter input into the AF-475 shell (Figure 38). The input is then sent to the second ASP (af475a.asp) which puts it into temporary data files and then queries the CI database for the list of Program Managers who can receive it. This list is then displayed to the client in a static HTML page containing a form with a drop down list (Figure 39). They can then select the recipient of the email. The third ASP (send475.asp) sends the temporary files to the selected PM and generates a response indicating the results (static HTML) of the attempt to email the data (Figure 40). Figure 41 shows the DFD for the TR input component indicating the implementation technologies.

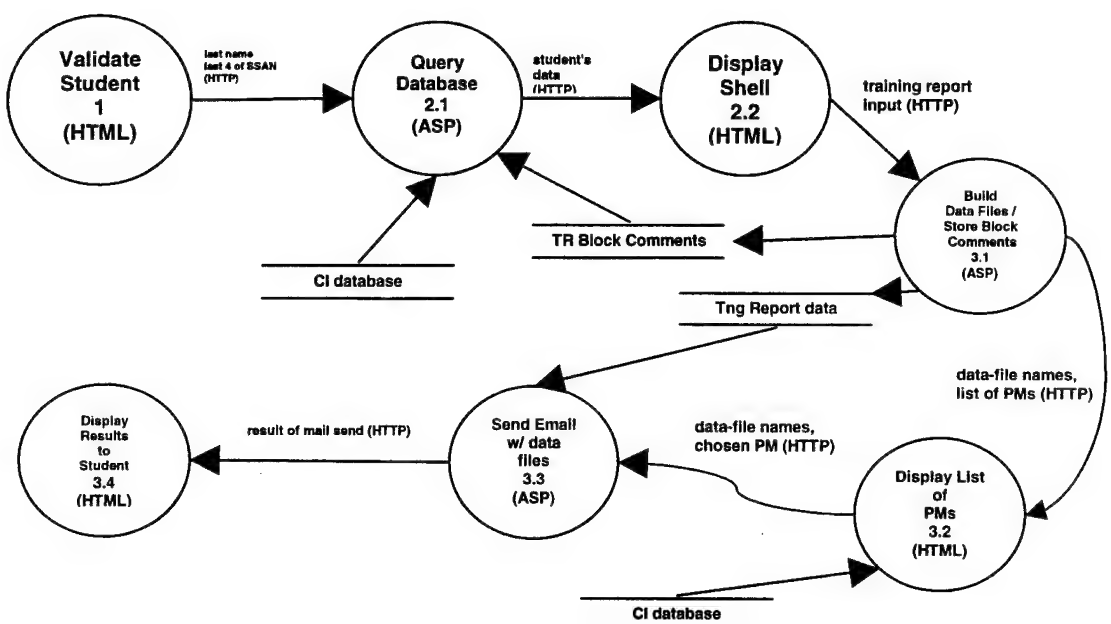

Figure 41. Component DFD with implementation technologies

4.2.4 Summary

In this chapter, it has been demonstrated how the methodology described in Chapter 3 can be applied to solve real-world web-database access problems. In accordance with the goals of this research, the three components developed show that the methodology detailed in Chapter 3 aids a developer in building components that provide more efficient and new ways to make use of existing relational data sources through the World Wide Web.

Chapter five summarizes the goals, methodology and results of this research and presents several areas in which further research efforts can be directed.

5 Findings and Conclusions

This research briefly examines the evolution of the client-server architecture and the emergence of the Internet and the World Wide Web in view of its impact on the client-server model. The benefits brought to the client-server model by the web and several difficulties with using the web in a client-server role are discussed in earlier chapters. A methodology to assist developers in choosing among competing Internet software technologies to provide web-based relational database access has been proposed (Chapter 3) and implemented (Chapter 4).

5.1 Findings

This section begins with an overview of the findings made during this research. Some specific issues surrounding the application of the methodology proposed in the research to the AFIT/CI test case are discussed. Also discussed are several findings relating to the use of several technologies that were applied in the test case.

5.1.1 Overview

In the analysis of AFIT/CI's operations, several distinct observations are made. The first and second observations deal with the external and internal client environment. The last observation pertains to the existing database management system architecture.

When analyzing the external client environment it was difficult to estimate the number of potential clients who may access data and information available to potential students. While the number of external clients currently enrolled in a CI program is easily obtainable from the CI program managers of each CI program, the number of clients who may be considering applying for a CI program is much more difficult. No data is kept on the total number of applications made to the various CI programs and, understandably, no record is kept of direct inquiries made by prospective students to program managers. In light of this difficulty, an estimate of prospective clients was made with the aid of the CI staff. We have only moderate confidence in

the actual estimate, though the order of magnitude is considered appropriate.

Not having more complete information on the potential size of external clients could result in an overloading of the web-server if the actual number of accesses eventually exceed the capacity of the server. Since AFIT is using Microsoft Internet Information Server (IIS) on a dedicated server, this should occur with low probability. However, periodically monitoring those components available to external clients can assist in preventing a server overload. This monitoring should include the collection of statistics such as the average number of daily or hourly hits on those pages, the average number of concurrent connections at any given time, and average number of distinct clients that access those components. Such information can be used periodically to determine the actual server load due to external clients. The site administrator or developer may then make adjustments as necessary.

Conversely, the internal client environment is quite homogeneous. All internal client computers are PC based and are running the Microsoft Windows 95 operating system. With the exception of a small number of clients, all internal users are using Microsoft Internet Explorer (3.0 or 4.0) as their web browser. This is a distinct advantage for the developer, since any components developed using Microsoft technologies such as client-side VBScript or ActiveX will be assured to perform predictably and well. Components for these clients can therefore be developed that work in close harmony with the operating system, enjoying the ability to perform file I/O and printing and to run at native OS speed. To ensure a homogeneous client configuration continues to exist, AFIT/CI should require that all internal clients use Microsoft Internet Explorer if possible.

In performing an analysis of the existing database architecture it was found that while the current Microsoft Access database was fulfilling current data needs, it has reached its ability to scale-up to further requirements. The database was designed and implemented prior to the emergence of the Internet or WWW and was designed for single-user, single-computer operations. As time progressed, the ability to access the data over a LAN was introduced and the

size of the database grew. The database is currently in excess of twenty megabytes of data. With the addition of web-related data such as email addresses, web page addresses, and persistent data, future web-based components such as on-line education plan maintenance and training report inputs will overburden the Access database. An appropriate recommended upgrade is described in Section 5.2.1.

Another area of concern was the independent naming of fields within database tables for the same data (semantically). This made it difficult to determine the correct attributes required to perform join operations. For example, the field to represent a civilian institution's supporting CBPO is called CIV_INS00CBPO_CODE in the CI_QUERY_CIVILIAN_INSTITUTION[30] table and called CBPO_CODE in the CI_QUERY_CBPO table. The absence of explicitly defined primary keys within tables, and tables that shared an obvious relationship that was not defined or had no relational integrity, also made the existing database analysis difficult. By not taking advantage of the most powerful aspects of relational databases, this not only affects the overall maintainability of the database, but also will hamper the performance of both local and web database operations.

While the CI database has met all requirements placed on it to present, the additional performance, storage, and access needs of the additional data that can result from web-based access can be realized by reevaluating and redesigning the database. Specific recommendations on the design are made in Section 5.2.1.

5.1.2 Issues regarding application of methodology and technologies applied

This section describes the results of implementing the three components that comprise the test case using the technologies that were chosen as a result of the methodology. Each subsection covers a specific issue and proposes one or more possible solutions to resolve the issue.

[30] While they are in fact tables and not queries, many tables have the word 'QUERY' in their name.

The overall application of the methodology to the test case proved to be very flexible. After the initial functional analysis, design, and implementation of the first component (School Search Component), the expected web server platform for the components was changed. The component was originally implemented as a Java servlet designed to run on a Java enabled web server. The change in web server platform had no impact on the choice of which type of technology that would be used (a web server process) to perform the data access. The only impact was that the component was rewritten as an ActiveX Server Page, which is the native web server processing method of IIS. The functionality and behavior of the component remained unchanged.

Several issues came to light during the implementation of the three components comprising the test case. They are the data security of the CI database, the need to perform data conversions on some student inputs, and the variations in how web browsers display standard HTML pages.

5.1.2.1 Data Security

In the analysis of the two web software components that would service current CI students, it was clear that an acceptable level of data security was necessary. While the data contained in the database is not classified, elements of it are covered by the Privacy Act of 1974. Therefore, measures were taken to ensure that only authorized clients could view a student's data.

In order to verify that a client is authorized to view a student record, he must correctly enter his last name and the last four digits of his Social Security Number before viewing data. While this is not considered to be an infallible method of security, it was deemed appropriate for the type of data being stored. In addition to the client authentication process, web clients make no live updates into the main CI database. Only files containing recommended updates are provided to the PM for approval and commitment.

The authentication for the two components that provide CI student functions (Personal Data Update, and TR Input) is implemented through an ActiveX Server Page (menu.asp) that

allows a client to log-in and then presents a menu of available functions to authenticated users. The student's last name and last four digits of their SSAN are passed as parameters to all components chosen from this menu. Upon leaving a component they reenter the menu application which ensures again that they are authentic. This is done to not only aid in retrieving their information, but to also ensure that they view only the data that pertains to them. While this data security plan is adequate, a better, more robust plan would incorporate the security features of the web server (IIS). Section 5.2.2 provides more detailed recommendations on implementing a data security plan for web clients.

5.1.2.2 Data Conversion to support forms processing

In implementing the Training Report Input Component (Chapter 4, Section 4.2.3) it was discovered that a data conversion of the student's TR input was necessary. The requirements specification stated that the output of the training report was to be sent to the program manager as an email attachment that could be saved and subsequently loaded into their automated forms processing package. The forms processing package in use by the program managers is Delrina FormFlow. In order to read data into the FormFlow AF-475 form (Training Report) a data file must be created. The data file could be in several forms including an ASCII file in which the data fields were comma-delimited. This was the chosen format due to the low level of difficulty in creating ASCII files and the increased portability of ASCII data files when using past, present, and future versions of FormFlow.

In order to determine the exact composition of the data file(s), FormFlow was used to create one from an existing file. The AF-475 form was loaded into FormFlow and every possible data field was filled in. The data was then exported to a comma-delimited ASCII file using FormFlow's export capability. In analyzing the files created in this process, it was discovered that two files were actually created. The first file was a header file that contained information on the specific data format required by the AF-475 form. The second file contained the comma-

delimited strings of actual data from the form. While the first file (header file) was based on the form and was static, the second file changed depending on the data in the form.

With this knowledge, the component was developed to create both the header and data file from the TR input entered by the student (client), and to attach both files to the email sent to the PM. The PM could then easily save the attachments and load the data using FormFlow. FormFlow's inherent capability to read in the comma-delimited data files is accomplished by reading in the header file to determine the organization of the data, followed by the data in the data file.

5.1.2.3 Variations in web browser interfaces

In analyzing the external client environment (Chapter 3, Section 3.2.2) the developer can encounter many incompatibilities between client browser versions that can affect the way a component functions or is displayed on the client's machine. Chapter 4 shows how the developer could overcome several of the incompatibilities by using web server processes that return standard HTML pages to the client instead of downloaded components.

While virtually all browsers support standard HTML, the way that they display the HTML-encoded information may vary between browsers. For example an HTML checkbox in Microsoft Internet Explorer appears as a circle that gets filled in when the box is checked. In Netscape 3.0 on the Solaris OS however, the same checkbox will appear as a button that can either be raised or depressed to indicate whether it is checked.

Another area in which the client browsers may differ is in their screen resolution. An HTML object can appear differently in browsers with different screen resolutions. For instance at one resolution an HTML form set for 50% of the screen will appear much wider on a screen that is at a higher resolution. If the developer chooses to set the form width in screen pixels instead, the form may appear noticeably smaller on the screen with the higher resolution that displays more pixels.

While web server processes are capable of obtaining and reacting to client browser properties, they only receive some of the properties that can affect how the client views a web page. While this approach has some merit a more comprehensive approach would be to include the capability for clients to personalize the way a site is displayed on their browser. Section 5.4.2 covers the concept of personalization in more detail and explains how it can be used to provide a more suitable client-interface.

5.2 Recommendations

Based on the knowledge gained from the background research and from designing and developing several web software components for AFIT/CI, the following recommendations are offered with regard to the existing database architecture and data security plan.

5.2.1 Existing CI Database

In order to prepare the AFIT/CI database for the increased processing demands required of web-accessible databases, and to take better advantage of the capabilities of a relational database, the following recommendations are offered:

- **Use Microsoft SQL server on the web server**. Microsoft BackOffice, which is the AFIT web server software platform, contains a powerful relational database server named Microsoft SQL server. SQL server has the capability to access databases created with Microsoft Access, and has a greater capacity for concurrent connections and a more powerful database engine than Access. The CI database can then be maintained in MS Access locally with a linked copy maintained on the web server using SQL server.

- **Reorganize the CI Database**. The CI database should be redesigned using an entity-relational design tool such as Logic Works ERWin. Using this tool, CI personnel can identify the entities and associated attributes of all key players in the CI arena. Particular attention should be paid to ensuring that each table has a primary key, that semantically

like data attributes are named alike in all tables in which they appear, and that all relationships between tables are well defined and incorporated into the schema. Table names and attribute names should only be long enough to correctly identify what they are. For instance, there is no semantic loss in changing CIV_INS00CBPO_CODE to CBPO_CODE. Referential integrity should also be a main concern in the database to ensure that data is consistent between tables. While the attribute names and relationships can be modified within the existing database without data loss, adjusting the number or contents of tables may require more complex data transformations. These changes will maximize the performance of the database management system and ease the overall task of database maintenance.

- **Addition of web related data to tables**. Attributes should be added to the appropriate tables for the web address (URL) and email address of each civilian institution and the email addresses of each CI student and CI staff member. These attributes will allow web-based applications to take advantage of the information on other web sites, and to automatically take advantage of email systems as an automated data transport mechanism. It will also allow prospective students to learn more information about a school from a school's web page.

5.2.2 Data security measures.

In order to provide flexible and more secure data security of the CI database (on the web server) the following recommendation is made.

Use NT and IIS security. Internet Information Server (IIS) can be made to use the user account information of Windows NT (the server OS) when providing access to web applications. Each CI student should be provided a userid and password upon entering a CI program. The Windows NT user permissions of those IDs could then be used to

control access to web components and database tables, and to maintain session information on connected clients. This would provide a data security mechanism that would be flexible and scalable.

5.3 Conclusions.

From the research conducted it is evident that there are many competing Internet software technologies that are capable of providing web-database access. However, from the results of applying the methodology provided in Chapter 3, it has been shown that by performing an analysis of the development environment and by functionally analyzing the components to be developed, that the developer can obtain all the necessary information required to make the appropriate choices of implementation technologies.

Through the development of several web software components for AFIT/CI, it was further shown that web-based database access can not only create new functions and uses for existing data, but can also improve existing operational functions and data utilization.

5.4 Future work

There are several areas of study related to this research that provide both interesting and important avenues for further research. They include the use of CORBA for client-side access to distributed data sources and the use of personalization in designing client interface components.

5.4.1 Using CORBA for distributed data access.

The movement to use applications with a CORBA interface has gained momentum in enterprises that must make use of large amounts of distributed data. While CORBA provides a powerful method to exchange data, it has seen little application on the client-side. This is due to the fact that the ORB classes required by such an application are not part of the standard Java package and must therefore be downloaded with the application to the client's machine. The

large amount of downloaded code reduces the performance of the client, especially on low bandwidth connections [Acker97].

In 1997 however, Netscape Communications released its new web browser named *Communicator*. Communicator includes a client-side ORB. If other web browser vendors follow suit, this advance could lead to smaller and more powerful database access applications that can use the ORB of the browser to access any distributed resources. By including the ORB in the browser, the amount of applications code that needs to be downloaded to the client is reduced, increasing client performance.

This advance opens the door for many client-side applications that could benefit from a CORBA capability, most notably data intensive ones. Consequently, there are many opportunities to evaluate downloaded CORBA applications as a means of providing access to distributed databases (Object Oriented and relational).

5.4.2 Varying server responses based on client platform configurations.

Section 5.2.1 discussed the difficulties in providing a standard user interface to clients, even when the web software can detect their specific browser version. Many current trade resources and periodicals are giving increasing attention to web-sites that are personalizable rather than reactive to client settings. For instance, the first time a client uses a site he can provide some information that lets the server know things about his computer that will allow it to tailor output more precisely to the client.

A client logging into a personalizable site for the first time might not only be checked to see what browser, but also may be asked questions regarding other settings. These settings may include the resolution at which the client's monitor is set and a color scheme that is pleasing to the client. Additionally, preferences on whether the client prefers applets or ActiveX controls, which client side scripts run best on their computer, which applications they wish to see have available to them, and a host of other possibilities may be stored.

This information on the client's preferences can persist in several ways. The web-site could write these preferences to a "cookie" that gets saved to the client's computer and gets sent with any HTTP request directed to that site. The web-server process can then read the cookie and deliver content to the user based on their preferences. An alternative method would be to save the client's preferences on the web-server. Upon entry to a site the client would then log-in, allowing the server to load their preferences for all pages sent to that client.

While personalization allows the client to ensure that they have a functional interface and assists the web server in determining client properties, its main drawback is that it violates many tenets of an open architecture such as developers of Internet applications desire. Developers would again have to write several versions of any downloaded applications (ActiveX and Java) or client-side scripts (JScript, JavaScript, or VBScript).

5.5 Remarks

While the Internet software technologies applied in this research have provided a significant contribution to the traditional client-server environment, new and enhanced technologies are developed constantly. It therefore remains a daily technical and intellectual challenge to remain abreast of this ever-changing field of computing to ensure that applications developed make the best use of current client and server technology.

Bibliography

[Acker97] Acker, Michael, L.,
An Examination of Multi-Tier Designs for Legacy Data Access, Thesis
Dayton: Air Force Institute of Technology, 1997

[Baker97] Baker, Sean; Cahill, Vinny; Nixon, Paddy.
"BRIDGING BOUNDARIES: CORBA in Perspective",
IEEE Internet (Sept-Oct 97): 52-57

[Chappell97] Chappell, David, Linthicum, David S., "ActiveX Demystified"
BYTE (September 1997): 56-64

[Cornell96] Cornell, Julie, *Building a Dynamic Web/Database Interface*, Thesis
Monterey: Naval Postgraduate School, 1996

[Curtis97] Curtis, David. "Java, RMI and CORBA." 1997, WWWeb,
http://www.omg.org/news/wpjava.htm, (15 Nov 97)

[Gesing97] Gesing, Ted, Schneider, Jeremy. JavaScript for the World Wide Web,
Berkeley: Peachpit Press, 1997

[Hamilton96] Hamilton, Mark A. "Java and the Shift to Net-Centric Computing.",
IEEE Computer (August 1996): 31-39.

[Harmon97] Harmon, Trevor, "JavaBeans vs. ActiveX"
Web Informant (July 1997) : 13

[Hayes97] Hayes, Frank, "Everybody get Web Happy"
Computerworld (February 97) : 44

[Hoffman97] Hoffman, Paul, "Paul Hoffman's Server Comparison Chart." WWWeb,
http://webcompare.internet.com/chart.html, (12 Nov 97)

[JavaScript97] "JavaScript Guide." 1997, WWWeb,
http://developer.netscape.com/library/documentation/communicator/jsguide4/index.htm

[JavaSoft97a] "JDBC Guide: Getting Started." 1997, WWWeb,
http://www.javasoft.com, (Jul 97)

[JavaSoft97b] "Java™ RMI Tutorial." 10 February 1997, WWWeb,
http://www.javasoft.com/ (Aug 97)

[JavaSoft97c] "The Java Servlet API", 1997, WWWeb,
http://jserv.javasoft.com/products/java-server/webserver/fcs/doc/servlets/api.html
(14 Aug 97)

[Kristula97] Kristula, Dave. " The History of the Internet." March 1997, WWWeb http://www.davesite.com/webstation/net-history.shtml (3 Dec 97)

[Microsoft97a] "Active Server Pages FAQ", 1997, WWWeb, http://www.microsoft.com/iis/guide/aspfaq.asp (18 Dec 97)

[Microsoft97b] "Understanding ODBC and OLE", 1997, WWWeb, http://www.microsoft.com/odbc/wpapers/odbcnole.htm (15 Nov 97)

[Morgan97] Morgan, Bryan, "CORBA meets Java" JavaWorld, (October 1997), WWWeb, http://javaworld.com/javaworld/jw-10-1997 (25 Oct 1997)

[Morrison97] Morrison, Michael, *Java Unleashed*, Second Edition. Indianapolis: Sams.net, 1997

[Mueller97] Mueller, John Paul, *ActiveX from the Ground Up*, Berkeley: Osborne , 1997

[NCSA97] "Common Gateway Interface." 1997, WWWeb, http://hoohoo.ncsa.uiuc.edu/cgi/overview.html (21 Nov 97)

[NWG97a] Network Working Group, RFC2068, "Hypertext Transfer Protocol -- HTTP/1.1", January 1997, WWWeb, http://www.cis.ohio-state.edu/htbin/rfc/rfc2068.html (3 Dec 97)

[NWG97b] Network Working Group, RFC2109, "HTTP State Management Mechanism", February 1997, WWWeb, http://www.cis.ohio-state.edu/htbin/rfc/rfc2109.html (4 Dec 97)

[OMG97a] "What is Corba?" 1997, WWWeb, http://www.omg.org/corba.htm (2 Dec 97)

[OMG97b] "Java, RMI, and CORBA.", OMG white paper, WWWeb http://www.omg.org/news/wpjava.htm

[Orfali97] Orfali, Robert; Harkey, Dan. *Client / Server Programming with Java and CORBA*, New York: John Wiley & Sons, 1997

[Perl97] "Perl for Win32 Frequently Asked Questions (FAQ)", 1997, WWWeb, http://www.perl.org/CPAN/ports/win95/FAQ (14 Jul 97)

[Pountain97] Pountain, Dick, Montgomery, John, "Web Components" Byte (August 97) : 56 - 68

[Rumbaugh91] Rumbaugh, James; Blaha, Michael; Premerlani, William; Eddy, Frederick; Lorensen, William. *Object Oriented Modeling and Design.* Englewood Cliffs: Prentice Hall, 1991

[Saadawi94] Saadawi, Tarek N., Ammar, Mostafa, H., El Hakeem, Ahmed
Fundimentals of Telecommunication Networks.
New York: John Wiley & Sons, 1994

[WWW96] "Hypertext Markup Language (HTML)." November 5, 1996. WWWeb,
http://www.w3.org/pub/WWW/Markup/ (15 Nov 97)

Vita

Captain Daniel L. DiPiro was born on 15 July 1965 in Ansonia, Connecticut. He graduated from Notre Dame High School, West Haven, Connecticut in 1983. He enlisted in the Army in 1985 as a Computer Programmer / Analyst. In 1987 he was selected for the Army ROTC program and attended St. Thomas Aquinas College to complete a Bachelor of Science Degree in Business Administration / Management Information Systems. He graduated Summa Cum Laude in 1990 and received a commission as a Lieutenant in the U.S. Army Signal Corps. After attending the Signal Officer Basic Course and the Teleprocessing Operations Officer Course (TOOCII), his first assignment was to the 97th Signal Battalion (CENTAG NATO) as a communications platoon leader.

In 1993 CPT DiPiro attended the Signal Officer Advanced Course and the Tactical Signal Operations Officer Course. He subsequently served as the Battalion Signal Officer for the 297th Military Intelligence Battalion, Ft. Gordon, Georgia until 1995. In 1995 he was chosen to command Delta Company, 63d Signal Battalion, Ft. Gordon, Georgia. Following his company command, CPT DiPiro attended the Combined Arms Services and Staff School. In 1997 he began attending the Air Force Institute of Technology. After completing the AFIT program in 1998, CPT DiPiro will be the Information Management Officer for the Directorate of Admissions, United States Military Academy at West Point.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	24 March 1998	Master's Thesis

4. TITLE AND SUBTITLE

Methodology for the Analysis and Design of Internet Software Components Providing Relational Database Access Through the World Wide Web

5. FUNDING NUMBERS

6. AUTHOR(S)

Daniel L. DiPiro, Captain, USA

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
2950 P Street
WPAFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT/GCS/ENG/98M-01

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFIT/CI
1st Lt Rick Sutter
2950 P Street
WPAFB, OH 45433-7765

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This work examines the application of Internet software technologies to provide access to remote relational databases via the World Wide Web. The research applies these software technologies to assist the Air Force Institute of Technology Civilian Institute Program in improving operations and student-to-staff communication.

An analysis of the existing Internet software technologies revealed several competing technologies capable of performing the same database access functions. The analysis further revealed weaknesses and inconsistencies in the existing AFIT/CI database. A methodology is proposed to assist in analyzing an existing development environment and in selecting among the competing technologies to provide the web-based database access. The methodology is applied to the AFIT/CI test case to demonstrate a technique of analyzing and designing web software components that will create new and improved uses for the existing CI database. Additional recommendations are also offered to improve the existing database operations.

The results of applying the methodology demonstrated that it effectively focuses the developer on the key areas of the development environment necessary to choose among competing software technologies. Additionally, the methodology was proven to be flexible in response to changes in implementation technologies.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Data Bases, Internet, Object Oriented Programming, Software Engineering	122
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18 298-102

USAPPC V1.00